UCLouvain

École polytechnique de Louvain

Learning through Optimization Programs

Author: **Brieuc PINON**
Supervisor: **François GLINEUR**
Readers: **Raphaël JUNGERS, Vincent WERTZ**
Academic year 2018–2019
Master [120] in Mathematical Engineering

Contents

Chapter 1

Introduction

The field of Machine Learning broadly addresses the problem of selecting a function or a model given samples of interest. Machine Learning helps to tackle problems of classification, regression or prediction tasks, it can also help with data analysis.

Machine Learning has a large literature devoted to variations of this problem. In this work we will focus on the classical supervised learning of a function given independent identically distributed samples of input-output pairs.

Our contribution is the exploration of the possibility to learn models described by optimization programs. More precisely, the model we consider is a function of the input given by

$$F_\theta(x) = \arg\min_y f_\theta(x, y), \tag{1.1}$$

where x is the input, y contains the prediction and θ denotes the parameters of the model.

Applications The domain of application is the learning in a supervised setting. Besides this broad domain specific applications that could profit from the development have been identified.

A possible specific application of this work is the development of optimization program components inside a larger model. This allows to make hybrid compositions such as neural networks with some layers corresponding to learned optimization programs.

There are already examples using optimization programs inside some proposed Deep Learning architectures in the literature (we refer to chapter 5). These propositions could profit from the tools developed here.

Another application of interest is the development of learnable theories using a language that can represent high-order structure (see chapter 6).

Probabilistic models A parallel can be done with probabilistic models, since probabilistic models [8, 20] and optimization programs' function both assign real numbers to possible outputs. The optimization program models perspective is reminiscent and can be an alternative to probabilistic models in some situations.

The main differences between the two approaches is the loss of probabilistic interpretation for optimization programs' objective function. The interpretation is accompanied by the constraints attached, the loss comes thus with new degree of freedom. In the case of optimization programs possible computational gains can be expected due to not having to enforce the respect of this interpretation in the assigned numbers.

Structure of the thesis We recognize our learning problem as a bilevel optimization problem. Consequently a presentation of the bilevel optimization field is done in chapter 2. While our problem is bilevel it is observed that the bilevel optimization literature does not gives us the necessary tools to address our problem.

In chapter 3 and 4, learning programs are designed and tested in the class of convex programs. From experimentation, a model is incrementally constructed based on observed limitations. Finally a practicable model is produced and tested.

Chapter 5 explores learning for non-convex programs. Non-convex programs pose intrinsic difficulties for learning, ideas are given to address these and some are tested. While no concrete positive result comes out of the study, a new research direction is given for further research.

Chapter 6 and 7 highlight further constructions around the tools developed notably with potential applications in Deep Learning and in the learning of declarative theories with high-order structure.

Chapter 2

Bilevel Optimization and Problem Formulation

Independently of the Machine Learning applications we will study a more general formulation of the problems we wish to solve is optimization problems defined using the arg min operator.

These problems are studied in the field of bilevel optimization that we present in this chapter.

2.1 Bilevel Optimization : definition and applications

A general bilevel programming problem is of the form [14]

$$\begin{aligned} \min_{x \in X, y} \quad & F(x, y) \\ \text{s.t.} \quad & G(x, y) \leq 0, \\ & y \in \arg\min_{y} f(x, y) \text{ s.t. } g(x, y) \leq 0, \end{aligned} \tag{2.1}$$

with $x \in \mathbb{R}^{n_1}$, $y \in \mathbb{R}^{n_2}$ and G, g are vector functions possibly encoding several constraints. A common denomination for x, F and G are upper-level variables, objective and constraints respectively. y, f and g are called lower-level variables, objective and constraints respectively.

Set X offers another possibility besides $G(x, y) \leq 0$ to encode constraints independent of the lower-level variables y in an explicitly way.

Note that while usual stochastic optimization problem can seem close to this problem, it is the optimal value of the lower-level optimization problem that is of importance for the upper-level problem in stochastic programming,

and not the variable achieving this minimum (min/arg min distinction). This difference is of importance for the mathematical properties and applicable algorithms (see next section).

Applications of bilevel optimization are present in a broad variety of fields [52]. While we refer to the just cited article for more references and specific examples, here is presented the variety of application domains. An informal descriptions of these domains and of the relevance of bilevel optimization in them is given.

Game theory In games where a leader can make an action posing the rules for a sub-game played with other players, called followers, a formulation with a bilevel program is indicated if the sub-game outcome can be posed as an optimization problem.

Examples include :

- In Stackelberg competitions where leading firm can pose an action from which a follower firm in competition act rationally based on it [47].
- Games with a regulator (leader) and regulated entities (followers), note that they are also a Stackelberg competition.

  Instances of this scheme can be found in environmental economics where a regulator has to pose rules for polluting industrials knowing their response to the chosen rules via an optimization problem simulating the problem on their side. The regulator on his side can choose to optimize a compromise between the generated pollution and the revenues [1, 53].
- Other instances of this regulator-regulated pairs appears in network optimization [33, 5]. Notably the Toll Setting Problem [10] where a regulator can pose tolls on a network, then based on it users can respond by adapting their path to minimize their cost. Both party having their objectives, what is the best possible toll dispatch for the regulator?
- Several applications with the same action-reaction logic also exist in defense applications [9]. A specific example is the positioning of missile interceptors [11].
- A group of common games in real life under the name of Principal-agent problem also follows the leader-follower logic [31]. Here the leader proposes a contract to a follower: employer-employee, politician-voters.

Chemical and Physical Industry Industrials who wants to optimize some process can be faced with bilevel problems. The lower-level problem correspond to the formalization of a physical principle such as entropy minimization, in order to obtain the output of a reactor for example [46]. The upper-level problem encode usual concerns found in the industry such as optimizing costs.

Another example is the optimal design of some structure (structural optimization) [24]. The upper-level can concern the stability, weight and cost of the structure while the lower level encode the potential energy minimization principle (i.e. the physics of the system).

Inverse Optimal Control Usual control consist in the determination of optimal action to control a system with respect to a cost function. However, in practice encoding a formalization of the informal objective can be cumbersome; involving notably parameters tuning with trials by hand.

Inverse optimal control aims at formalizing the problem at hand at his true level. For example it can consists in the optimization of an objective function giving a behavior after optimal control, the upper-level objective being written on the basis of the obtained behavior. Bilevel-optimization is thus a natural framework for these problems [28, 55].

Machine Learning: hyper-parameter optimization and meta-learning Machine Learning frequently involve the optimization of parameters of a function on a training set under some loss measure. One hopes by doing this to also implicitly reduce the loss on unseen examples, in another word generalize.

Practitioners of Machine Learning frequently separate randomly the available data in three sets: the training set, the validation set, and the test set. Given this separation, the parameters are fixed by fitting the training set given the value of the hyper-parameters; the hyper-parameters are chosen to maximize a fitting performance of the parameters on the validation set; end the final performance is evaluated on the test set. Note that there exist many variants to this procedure.

This procedure involve implicitly a bilevel optimization problem. We develop this view.

The optimization problem to choose the parameters, and so the function, has to encode preferences among the set of functions [38].

Such preferences are typically controlled via hyper-parameters. The determination of the hyper-parameters is done by constructing a bilevel optimization problem where the upper-level loss is the loss on a validation set

independent of the training set for the prediction function determined by the hyper-parameters, and where the prediction function for given hyper-parameters is the one found by the lower-level problem based on the training set.

Note however that most hyper-parameter optimization methods ignore this structure and instead use general black-box optimization algorithms. Exceptions to this remark can be found in [6, 7, 54, 36].

Another application closely linked to hyper-parameter optimization is meta-learning. Meta-learning learn a learning algorithm that perform well for a predefined class of learning problems. The technical difference with respect to bilevel hyper-parameter optimization is that the hyper-parameters are shared across several problems, see [18] on the subject.

2.2 Bilevel Optimization: Methods

In this section information about the existence of a solution are given and methods devised to handle bilevel problems are presented. The presentation is intended to be representative of the bilevel optimization literature. The precise class of programs used in this work and methods devised for them will be presented in more details further in the report.

2.2.1 Existence of solution

For general bilevel optimization problem the existence of an optimum is guaranteed under continuous and boundness assumptions on the involved functions.

A classic result is [52]

Theorem 1. *If the functions F, f, G, g are sufficiently smooth, the constraint region $\{(x, y)|G(x, y) \leq 0, g(x, y) \leq 0\}$ is non-empty and compact, and the Mangasarian-Fromowitz constraint qualification holds at all points, then the problem has an optimum.*

For mixed-integer bilevel problems under compactness conditions one can assure that the continuous-continuous, discrete-continuous and discrete-discrete linear cases admits an optimum, as studied in [57]. The continuous-discrete linear case is more problematic [52].

Other cases can be studied, for example in our case if there is non-unicity of the optimum in the lower-level problem, the upper-level can choose the answer that improves his objective. This is called the optimistic position

in the literature, a pessimistic position can also be devised for which other conditions for the existence of an optimum apply.

Discreteness associated to non-linearity can also be involved in practice.

See [15] for further information on optimum existence guarantees.

2.2.2 Methods

Bilevel optimization is algorithmically complex; to show it the restricted and classic case of bilevel problems with linear objectives and constraints is considered.

A short proof of NP-Hardness in this case can be given using the equivalence between the binary constraint on variable x and the upper-level constraint $y = 0$ associated to the lower-level optimization problem

$$y \in \arg\min -y \text{ s.t. } y \leq x,\ y \leq 1 - x$$

[14].

It is know that it is moreover strongly NP-Hard [22] and that under P$\neq$NP there does not exists a polynomial time algorithm that can approximately solve the problem [16, 27].

Despite these theoretical results multiple approaches to the problem have been designed. Here is an overview of the ideas based on the following reviews [14, 52].

Single level reduction with KKT

If the lower level is convex, sufficiently smooth and satisfies simple conditions, like the existence of a Slater point, one can replace the lower-level problem by usual algebraic conditions using the KKT conditions.

Applied to a linear bilevel problem, the KKT complementary conditions of the lower-level problem produce bilinear constraints that can be handled in a mixed-integer framework with branch and bound method for example [2].

A similar idea can be adapted for bilevel linear-quadratic and quadratic-quadratic problems [3, 17].

Descent methods

Here a descent direction on the upper-level variables is computed taking into account the constraints, notably those on the lower-level ones. The descent direction is then used to make an iterate decreasing the objective value.

The computation of such a descent direction is a challenge. Several methods have been designed with different area of applications. Those method include: approximation of the gradient assuming an implicit function from the upper-level variables to the lower-level ones $y(x)$ [30]; reducing the steepest descent search to a linear-quadratic bilevel problem [45].

Penalty Function methods

By substituting the hard-constraint for penalty terms in the objective the complex originally constrained problem can be changed for an unconstrained one. However the equivalence is not perfect and a succession of penalized problems have to be solved to solve the original problem.

The exchange of constraints for penalty function can be made at several places in the upper-level and/or lower-level constraints [48], but also after a single-level reduction on the KKT conditions [49].

Trust-region methods

Trust-region methods create at each iterate a simpler model of the optimization problem around the current point. They also associate to this model a region of trust. This simpler problem is then solved in the trust region to yield the next point.

An example in bilevel optimization of this method when the upper-level constraints only depend on the upper-level variables is to approximate the lower-level problem with a quadratic problem and to linearize the upper-level problem [13].

Evolutionary approaches

Genetic Programming (GP) and Particle Swarm Optimization (PSO) were both proposed to solve bilevel problems. They are attractive by their capacity to handle difficult problems with nearly no profitable structure.

They can be used in several ways: applied only at upper-level keeping a specialized algorithm at the lower-level [37] or at both levels [35]; applied on a single-level reduction of the problem [23]; or with meta-modelling methods that are described next.

With meta-modeling one wants to avoid the computing cost of the evaluation of a complex function by replacing it by an approximation simpler to evaluate.

In bilevel algorithms there are several choices of functions to approximate that make sense: approximate the minimizer set of the lower level problem given the upper-level variables [51]; approximate the optimal objective value

of the lower-level problem given the upper-level variables [58]; approximate the lower-level optimization problem by another one for which a faster specialized method exists [52].

Methods for mixed-integer bilevel programs

Specialized methods include adaptations of branch-and-bound [39]; implicit enumeration scheme [4]; benders decomposition [55]. However these methods does not scale to large problems [52].

2.3 Problem formulation

Using the structure offered by bilevel programming we formalize our learning problem. This is to the best of our knowledge a new application area for bilevel programming.

2.3.1 Supervised Learning

The basic supervised learning problem in Machine Learning consists in the estimation of a function g given a set of input/output pairs $\{(x_i, y_i = g(x_i)\}$. The function, also called model, can then be used on unseen data to estimate an evaluation of g.

There are lots of restricted classes for this problem: types of function binary/classification/continuous, number of examples, structure of the input, ...

The definition of an estimation performance can also be subject to variations. For example, in the continuous output case the mean square error can be used, or the accuracy in a classification task. In any cases the goal is to maximize this performance on unseen data. A classic hypothesis is to consider that available data follow the same probabilistic distribution as the unseen data.

Supervised learning has applications in image processing, natural language processing (speech and text), and medical diagnosis for example.

Functions class

The class of functions of interest in our search is

$$F_\theta(x) = \arg\min_{y \in S} f_\theta(x, y), \tag{2.2}$$

where x is the input, y is the optimized variable possibly constrained in set S (independent of X), and θ denotes the parameters of the model. We will concentrate our developments on input represented as reals vectors.

Formally F_θ is a function returning a set of elements, in our work we will make sure that this set contain one and only one element. We will thus make the notational simplification to confound the element and the set.

A trick we will use is to compose our model with the soft-max function in classification tasks. The soft-max function[1] takes $y \in \mathbb{R}^n$ as input and returns

$$\frac{1}{\sum_{i=1}^{n} e^{y_i}} \begin{pmatrix} e^{y_1} \\ \vdots \\ e^{y_n} \end{pmatrix}. \tag{2.3}$$

The final output can then be considered as a probabilistic distribution on the classes since his sum equal 1.

Another change of interest is to only keep part of the optimal vector variable y^* as output, letting the other dimensions play the role of hidden-variables. Another equivalent way to write this is by renaming the hidden-variables z giving

$$F_\theta(x) = \{y | \, (y, z) \in \arg\min_{(y,z)\in S} f_\theta(x, y, z)\}. \tag{2.4}$$

Adding hidden-variables is of importance, as it can allows one to describe f and S with fewer terms in some case. This argument is formalized on the mathematical programming side for linear program with polyhedral projections on subset of variables [32].

Learning Problem

In Machine Learning, learning algorithms are frequently posed as optimization problems. To learn these models a loss and an associated learning/optimization procedure are devised.

Specifically, for our supervised Machine Learning problem we have, with (x_i, y_i) the training input-output pairs (see the above introduction), L a loss measure, and R a regularization cost on the parameters

$$\min_\theta \sum_{i=1}^{N} L\left(y_i, \left[F_\theta(x_i) = \arg\min_{\hat{y}_i} f_\theta(x_i, \hat{y}_i)\right]\right). \tag{2.5}$$

[1] Another soft-max function will be used in this work. They are not used in the same way and can be distinguished from context. (The names are took from the literature.) Another proposed name is soft-argmax for this one.

An example of loss L for classification tasks that will be used jointly with the soft-max composition is the cross-entropy loss

$$L\left(y_i, F_\theta(x_i)\right) = -\log F_\theta(x_i)_{y_i}. \tag{2.6}$$

This problem possess the expected bilevel programming structure.

2.3.2 Unsupervised Learning

The goal of unsupervised learning is to learn a distribution or some properties of it from a set of samples $\{x_i\}$. This can be concretized by asking the learner to get a declarative model M from the data and taking into account a priori knowledge on the model.

A model can be used to answer questions on possible x_i like probabilities of occurrence in probabilistic models.

Designing a declarative model from data with unsupervised learning in our optimization program framework will not be studied in this work.

Our framework, when compared with respect to others like the probabilistic framework, is not justified by being an usual way to formalize problems, the search in this framework has begun for computational reasons.

This partially leads to the fact that there is no natural straightforward way to formalize the unsupervised learning problem in our case, which is still in contrast with probabilistic models.

We postulate that potential unsupervised learning with optimization programs should be on the basis of computational considerations in presence of no other clear desirable properties.

Chapter 3

Learning through Linear and Convex Quadratic Programs

We begin our exploration of program types for learning. The presentation of tests follows the research chronologically. For each type, limitations are observed and push the next proposition and associated experiments.

A seen in the previous chapter on bilevel optimization, the problem we wish to solve is computationally hard. Moreover, taking into account the usual scale of machine learning applications, the proposed implementation must be conceived with care for both optimization levels.

For the upper-level optimization, our devised optimization algorithms are based on local improvements, with the help of an heuristic direction such as a gradient for the parameters. These methods have the advantage to be scalable to large datasets, using their stochastic version (see below). The proposed class of programs must come with such an heuristic direction of search.

An example of stochastic version is the stochastic gradient descent algorithm (SGD). Assume a differentiable objective of the form

$$\sum_{i=1}^{n} f_i(x). \tag{3.1}$$

Stochastic gradient descent consist in applying successive iterates of the gradient descent algorithm of each f_i individually. The potential computational gain is to make small improvements without the need to evaluate the full sum.

Numerous evaluations of the program/lower-level optimization are expected, thus the program's class must admit a fast algorithm.

Linear programming is a natural choice to begin our exploratory work,

since optimization programs in this paradigm are easy to solve and can encode a large variety of problems that occur in practice.

However, our experience suggests that it is not the correct class of programs for our problem, and we extend it to quadratic programs in order to address one encountered problem. This extension will in turn reveals itself to be insufficient.

This study, while not directly fruitful, allows us to set in evidence (and has allowed us to discover) some properties that our programs should satisfy in order to be learnable. This knowledge will be used in the next chapter that will study other convex programs propositions.

3.1 Linear Programs

3.1.1 Model

We first consider this general formulation

$$\begin{aligned} F_{A,b,c}(x) = &\arg\min_{y} c^T y \\ &\text{s.t.} \\ &A \begin{bmatrix} x \\ y \end{bmatrix} \leq b \end{aligned} \tag{3.2}$$

Hence parameters $\theta = A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m, c \in \mathbb{R}^n$.

This general formulation pose problems:

- How one can ensure a non-empty domain?
- How one can ensure the problem is bounded?

We answer them by proposing a restriction.

To ensure a non-empty domain we optimize a piece-wise linear function determined as a maximum over a collection of linear functions. To ensure the existence of an optimum we constrain the variables in a box.

A first description of this program is, with A_i, B_i, c_i the rows and elements of the respective objects,[1]

$$\begin{aligned} F_{A,b}(x) = &\arg\min_{y} \max_{i=1,\dots,m} \{A_i y + B_i x + b_i\} \\ &\text{s.t.} \\ &\mathbf{-1} \leq y \leq \mathbf{1}. \end{aligned} \tag{3.3}$$

[1] $\mathbf{0}$ or $\mathbf{1}$ denote a vector of all zeros or respectively of all ones.

An auxiliary scalar variable t is added to encode the epigraph and we obtain the following linear program,

$$\begin{aligned} F_{A,b}(x) = &\arg\min_{y;t} t \\ &\text{s.t.} \\ &Ay + Bx + b \leq \mathbf{1}t \\ &\textbf{-1} \leq y \leq \mathbf{1}. \end{aligned} \tag{3.4}$$

Note that as there are no guarantees on the unicity of the optimum in this formulation, we make the assumption that non-unicity does not occur in practice for now.

3.1.2 Search Direction

The function $F_{A,b}(x)$ admit a first order approximation for points of the domain where the solution is locally unique and where there are locally no changes in the active constraints set for the optimal solution. For our developments we assume that the active constraints have linearly independent gradients.

In that case for our formulation 3.4 the optimum is locally given by

$$\begin{bmatrix} y^* \\ t^* \end{bmatrix} = \left(\begin{bmatrix} A & -\mathbf{1} \\ Id^{n\times n} & \mathbf{0} \\ Id^{n\times n} & \mathbf{0} \end{bmatrix}_{\mathbf{act}} \right)^{-1} \begin{bmatrix} -Bx - b \\ \mathbf{1} \\ -\mathbf{1} \end{bmatrix}_{\mathbf{act}}, \tag{3.5}$$

where $\begin{bmatrix} \vdots \end{bmatrix}_{\mathbf{act}}$ selects the rows corresponding to active constraints at the optimum.

The optimum can thus be obtained by differentiating through the matrix inverse operator. Let's use this to obtain an heuristic search direction.

The Jacobian of the inverse of a matrix $\mathbf{M}$ is a 4th order tensor given by [42]

$$\frac{\partial \left[M^{-1}\right]_{i,j}}{\partial M_{k,l}} = -\left[M^{-1}\right]_{i,k} \left[M^{-1}\right]_{l,j}. \tag{3.6}$$

Another way to give it is, with $M : t \to M(t)$ a function this time,

$$\frac{dM(t)^{-1}}{dt} = -M(t)^{-1} \frac{dM(t)}{dt} M(t)^{-1}. \tag{3.7}$$

Let us note the scalar objective function on the linear program output

$J(z)$. To derive the expression for $J\left(y^*\right)$, we note also

$$G = \begin{bmatrix} A & -\mathbf{1} \\ Id^{n\times n} & \mathbf{0} \\ Id^{n\times n} & \mathbf{0} \end{bmatrix}_{\mathbf{act}}, \tag{3.8}$$

$$d = \begin{bmatrix} -Bx - b \\ \mathbf{1} \\ -\mathbf{1} \end{bmatrix}_{\mathbf{act}}. \tag{3.9}$$

The gradients of interest are [2]

$$\begin{aligned} \nabla_G J\left(y^*\right) &= -\left(\begin{bmatrix}\nabla_z J\left(y^*\right) & 0\end{bmatrix} G^{-1}\right)^T \left(G^{-1}d\right)^T \\ &= -\left(\begin{bmatrix}\nabla_z J\left(y^*\right) & 0\end{bmatrix} G^{-1}\right)^T \begin{bmatrix} y^* \\ t^* \end{bmatrix}^T. \end{aligned} \tag{3.10}$$

From which we can extract $\nabla_A J(y^*)$ and, for the right term,

$$\nabla_d J\left(y^*\right) = \begin{bmatrix}\nabla_z J\left(y^*\right) & 0\end{bmatrix} G^{-1} \tag{3.11}$$

from which we can recover the gradient with respect to b and B (post multiplication by x) can be given.

Notice that, from a numerical point of view, no explicit matrix inversion is required but only few linear system solutions.

3.1.3 Experimentation

We try to make a regression of a simple function.

For the evaluation of the lower level program any linear programming solver can be used.

The used loss function is the usual quadratic difference between the model's prediction and the true value.

The optimisation (learning) algorithm used is simply a gradient descent where the used gradient is the developed search direction (which is nearly everywhere the true gradient). At each iteration the step is chosen to give an approximate minimizer along the search direction.

To test this first idea, we attempt to learn the binary XOR function, $\oplus$. It has the advantage to offer the possibility to have a visualization of the constructed program evaluations, allowing us to understand the inner working and drawbacks of our proposition.

[2] We take the row notation for gradient w.r.t. vectors, for gradient w.r.t. matrices the gradient has the same form as the matrix itself, and $M^{-T} = \left(M^{-1}\right)^T$.

Several numbers of planes (constraints) where used, the conclusions from this experimentation holds independently from this number, we note the existence of a perfect solution for three planes (and no perfect solution for only two planes).

The solution with three planes consists in: one plane $A_1 = -1$, $B_1 = [4, 4]$, $b_1 = -2$ which is dominant for inputs $(1, 0)$ and $(0, 1)$; one plane $A_2 = 1$, $B_2 = [0, 0]$, $b_2 = 0$ which is dominant for the input $(0, 0)$; and finally $A_3 = 1$, $B_3 = [8, 8]$, $b_3 = -8$ which is dominant for input $(1, 1)$.

Observation

There are zones in the search space where the gradient is zero, we will call them dead-zone. They happen in the situation visually presented in Figure 3.1. As one can see on the Figure, as soon as the optimum activate a $y \leq 1$ or

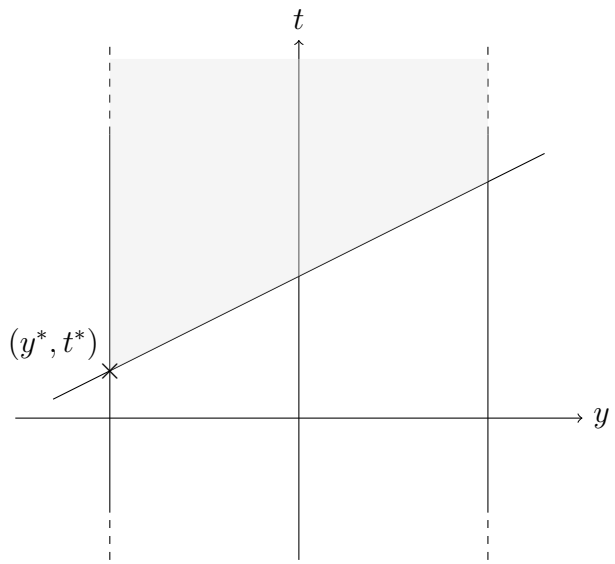

Figure 3.1: visualization of a dead-zone case

$y \geq -1$ constraint, any local move of the other constraints does not change the value of y^*.

In the situation where the value wanted for y^* is 1, a classic local optimizer based on the gradient, as the one used here, and starting with the situation on the Figure 3.1 does not reach this simple objective.

This problem inspired quadratic programs that are presented in the next section.

3.2 Quadratic programs

As said, earlier, the dead-zones problem inspired a quadratic program formulation. This new form substitutes the static linear box constraints for a quadratic norm term implying the existence and uniqueness of the optimum.

3.2.1 Model

The model is now

$$\begin{aligned} F_{A,b}(x) = &\arg\min_{y;t} t + \frac{1}{2}\|y\|_2^2 \\ &\text{s.t.} \\ &Ay + Bx + b \leq \mathbf{1}t. \end{aligned} \tag{3.12}$$

The program can be seen as a unconstrained problem where the objective is a maximum of linear terms plus a convex quadratic term.

It is strongly convex, the minimum is thus guaranteed to be unique in comparison with the previous linear program. Its existence is preserved also by strong convexity.

3.2.2 Search direction

In a similar way to the previous development for the linear program, a search direction equal to the gradient when there is locally no change in the active constraints set for the optimum. Also we assume linear independence between the active constraints gradients.

Suppose that there is locally no change in the active constraints set for the optimum, then locally the optimum is given by

$$\begin{aligned} \begin{bmatrix} y^* \\ t^* \end{bmatrix} = &\arg\min_{y;t} t + \frac{1}{2}\|y\|_2^2 \\ &\text{s.t.} \\ &\begin{bmatrix} A & -\mathbf{1} \end{bmatrix}_{\mathbf{act}} \begin{bmatrix} y \\ t \end{bmatrix} = \begin{bmatrix} -Bx - b \end{bmatrix}_{\mathbf{act}}. \end{aligned} \tag{3.13}$$

Here again $\begin{bmatrix} \vdots \end{bmatrix}_{\mathbf{act}}$ denote the selection of the rows corresponding to active constraints at the optimum in the original problem 3.12.

This is a constrained quadratic problem. We recognize two formulations allowing us to derive the wanted search direction.

Formulation using weights

The first formulation turns the problem into a least-squares problem in which the constraint are enforced with large scaling of the associated terms [19].

$$\min_{y,t} \left\| \begin{pmatrix} \frac{1}{\sqrt{2}} Id^{n\times n} & \mathbf{0} \\ \sqrt{\frac{M}{2}}[A & -\mathbf{1}]_{\mathbf{act}} \end{pmatrix} \begin{bmatrix} y \\ t \end{bmatrix} - \begin{bmatrix} \mathbf{0} \\ \sqrt{\frac{M}{2}} \left[-Bx - b\right]_{\mathbf{act}} \end{bmatrix} \right\|_2^2 + t, \tag{3.14}$$

where M is chosen sufficiently large to enforce the equality constraints in 3.13 but small enough to preserve us from devastating numerical errors. Note that the solution of this problem converges to the true solution with $M \to +\infty$ [19].

With this version it is possible to compute a gradient by differentiating through the associated analytical solution.

Analytical solution To write the analytical solution and the parameters' gradient two simplifying notations are used: the first is $Q = \begin{bmatrix} A & -\mathbf{1} \end{bmatrix}_{\mathbf{act}}$; and the second $d = \left[-Bx - b\right]_{\mathbf{act}}$. With it the minimization problem 3.14 can be rewritten

$$\min_{y,t} \frac{1}{2} \begin{bmatrix} y \\ t \end{bmatrix}^T \underbrace{\left(\begin{bmatrix} Id^{n\times n} & \mathbf{0} \\ \mathbf{0}^T & 0 \end{bmatrix} + MQ^TQ \right)}_{\stackrel{\text{def}}{=} P} \begin{bmatrix} y \\ t \end{bmatrix} + \left(-MQ^Td + \begin{bmatrix} \mathbf{0}^n \\ 1 \end{bmatrix} \right)^T \begin{bmatrix} y \\ t \end{bmatrix}, \tag{3.15}$$

for which the analytical solution is

$$\begin{bmatrix} y^* \\ t^* \end{bmatrix} = P^{-1} \left(MQ^Td - \begin{bmatrix} \mathbf{0}^n \\ 1 \end{bmatrix} \right). \tag{3.16}$$

Gradient The gradient of a function $J(z)$ based on the output, $J(y^*)$, with respect to Q is [3]

$$\begin{aligned} \nabla_Q J(y^*) = -MQ \left(\begin{bmatrix} y^* \\ t^* \end{bmatrix} \nabla_z J(y^*) P^{-1} + \left(\nabla_z J(y^*) P^{-1} \right)^T \begin{bmatrix} y^* \\ t^* \end{bmatrix}^T \right) \\ + Md \nabla J_z(y^*) P^{-1}. \end{aligned} \tag{3.17}$$

With respect to d it is

$$\nabla_d J(y^*) = M \nabla_z J(y^*) P^{-1} Q^T. \tag{3.18}$$

[3] Recall: we take the row notation for gradient w.r.t. vectors; for gradient w.r.t. matrices the gradient has the same form as the matrix itself; and $M^{-T} = \left(M^{-1}\right)^T$.

From which the gradient w.r.t. $\left[B\right]_{\mathbf{act}}$ is

$$\nabla_{\left[B\right]_{\mathbf{act}}} J(y^*) = -MQP^{-T} \left(\nabla_z J(y^*)\right)^T x^T. \tag{3.19}$$

Formulation using the Lagrangian

Another possibility is to solve 3.13 using a Lagrangian function [19].

We pose as for the previous development $Q = \left[A \quad -\mathbf{1}\right]_{\mathbf{act}}$ and $d = \left[-Bx - b\right]_{\mathbf{act}}$.

In our case the Lagrangian is

$$L(x; \lambda) = \frac{1}{2} \|y\|_2^2 + t + \lambda^T (d - Qx). \tag{3.20}$$

The optimality conditions becomes simply a quadratic problem, with m the dimension of the linear subspace spanned by the active constraints' gradients[4],

$$\begin{pmatrix} \begin{bmatrix} Id^{n\times n} & \mathbf{0}^n \\ (\mathbf{0}^n)^T & 0 \end{bmatrix} & -Q \\ Q & \mathbf{0}^{m\times m} \end{pmatrix} \begin{bmatrix} y \\ t \\ \lambda \end{bmatrix} = \begin{bmatrix} \mathbf{0}^n \\ -1 \\ d \end{bmatrix}. \tag{3.21}$$

Since the problem 3.13 is convex and admits a Slater point the KKT conditions are necessary and sufficient.

An analytical solution and a gradient can be found in a similar way to the previous development from this point.

3.2.3 Experimentation

The experimentation setting is identical to the one used for the linear programming approach except that a quadratic programming solver is used to evaluate the program, and that the search direction is changed to the one just developed.

Observation

The learning of the XOR function is easier w.r.t. the linear programming formulation. Unfortunately, it appears it is not possible to learn some more complex functions.

While the previously described problem with the box constraints has disappeared, another problem of zero-gradient stays.

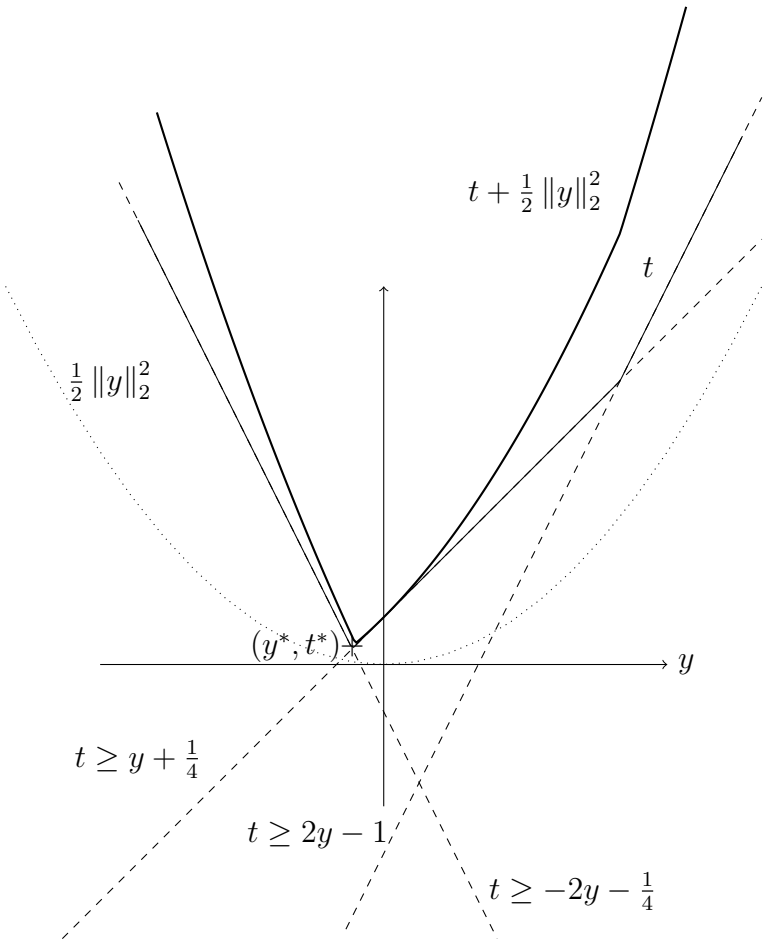

Figure 3.2: situation with inactivity of a plane for the quadratic programming formulation

This problem appears in situations similar to the one represented on Figure 3.2. In this situation one can observe that the plane $t \geq 2y - 1$ is not active at the optimum. Consequently the local influence of the hyper-plane's parameters onto the optimum is locally zero, and so is the related part of the gradient.

Thus, during the optimization planes can be implicitly discarded by our algorithm on the basis of their inactivity.

To address this problem two directions of research are considered, both relying smooth strongly convex programs.

[4]We are still using the assumption that all active constraints' gradient are linearly independent.

Chapter 4

Learning through Convex Programs

As noted, the problems encountered for the previous class of programs push us to test other solutions. Notably to extend our research to smooth strongly convex programs.

This class of optimization is of interest to us for several reasons.

- A global optimum exists and is unique.
- It is fast to evaluate with global optimality guarantees.
- If additional smoothness assumptions are satisfied a gradient with respect to the parameters is well-defined.
- We need to introduce some non-linearity but without hard-constraints giving non-informative local gradient.

The gradient mentioned in the third reason above will be given next. This gradient will be the basis of all the learning algorithms for the rest of the chapter.

These reasons are also satisfied by "pure" convex quadratic programs (i.e. without linear constraints). However, those can only represent a subset of linear functions since their evaluation can be done by multiplying the input by the matrix inverse, which is a linear operator.

This lack in expressiveness is judged to limiting for the class of programs to be useful.

All the experiments are reported at the end of the chapter.

4.1 Gradient of Smooth Strongly Convex Programs' Parameters

One is interested in the differential of

$$z^*(\theta) = \arg\min_z f_\theta(z). \tag{4.1}$$

Here we work in a general case, the dependence with a potential input in our machine learning application is implicitly coded in f.

Notation: $f'_\theta(z), f''_\theta(z)$ are the gradient and hessian w.r.t. z at the point (z, θ).

Our answer is developed for these conditions:

- f is a $\mathcal{C}^2$ function w.r.t. z for all θ, and the hessian respects $\forall \theta, z \quad 0 \prec \mu Id \preccurlyeq f''_\theta(z)$ (i.e. $\mu > 0$);
- $f'_\theta(z)$ is $L-$Lipschitz continuous w.r.t. θ;
- $f'_\theta(z)$ is differentiable w.r.t. θ;
- $f''_\theta(z)$ is differentiable w.r.t. θ.

Note that since differentiability is a local notion it is sufficient for these conditions to hold only locally. In a more formal way, the adaptation is that for any point and function of interest one can restrict the function on an open set containing the point and for which these conditions hold.

Theorem 2. *Under these conditions the following holds*

$$\nabla_\theta z^*(\theta) = -\left[f''_\theta(z^*(\theta))\right]^{-1} \nabla_\theta \left[f'_\theta(z)\right](z^*(\theta)). \tag{4.2}$$

Proof. The first and last conditions imply that $f''_\theta(z)^{-1}$ always exists and is differentiable w.r.t. θ. Thus, with the third condition, the Newton step $-\left[f''_\theta(z)\right]^{-1} f'_\theta(z)$ is differentiable w.r.t. θ.

From the first and second conditions, $z^*(\theta)$ is $(2\frac{L}{\mu})-$Lipschitz continuous w.r.t. θ since

$$\|z^*(\theta) - z^*(\theta + d\theta)\| \leq 2 \left\| f''_{\theta+d\theta}(z^*(\theta))^{-1} \right\| \left\| f'_{\theta+d\theta}(z^*(\theta)) \right\| \leq 2\frac{L}{\mu}, \tag{4.3}$$

by the quadratic lower bound.

Now we are able to prove that the expression $-\left[f''_\theta(z^*(\theta))\right]^{-1} \nabla_\theta \left[f'_\theta(z^*(\theta), \theta)\right]$ exists and is the gradient of $z^*(\theta)$ w.r.t. θ.

The existence is directly implied by the assumptions and the previous results.

To prove that it is indeed the differential of $z^*(\theta)$, one has to show that the following limit evaluates to zero,

$$\lim_{d\theta\to 0} \frac{\left\| z^*(\theta+d\theta) - z^*(\theta) + \left[f''_\theta(z^*(\theta))\right]^{-1} \nabla_\theta \left[f'_\theta(z^*(\theta))\right] d\theta \right\|}{\|d\theta\|}. \tag{4.4}$$

For that we will use the newton step $N_s = -\left[f''_{\theta+d\theta}(z^*(\theta))\right]^{-1} f'_{\theta+d\theta}(z^*(\theta))$. A manipulation gives,

$$\begin{aligned}
&\lim_{d\theta\to 0} \frac{\left\| z^*(\theta+d\theta) - z^*(\theta) + \left[f''_\theta(z^*(\theta))\right]^{-1} \nabla_\theta \left[f'_\theta(z^*(\theta))\right] d\theta \right\|}{\|d\theta\|} \\
&= \lim_{d\theta\to 0} \frac{\left\| z^*(\theta+d\theta) - z^*(\theta) - N_s + N_s + \left[f''_\theta(z^*(\theta))\right]^{-1} \nabla_\theta \left[f'_\theta(z^*(\theta))\right] d\theta \right\|}{\|d\theta\|} \\
&\leq \underbrace{\lim_{d\theta\to 0} \frac{\|z^*(\theta+d\theta) - z^*(\theta) - N_s\|}{\|d\theta\|}}_{L_1} + \underbrace{\lim_{d\theta\to 0} \frac{\left\| N_s + \left[f''_\theta(z^*(\theta),\theta)\right]^{-1} \nabla_\theta \left[f'_\theta(z^*(\theta))\right] d\theta \right\|}{\|d\theta\|}}_{L_2}.
\end{aligned} \tag{4.5}$$

Quantity L_2 is equal to 0 since $-\left[f''_\theta(z^*(\theta))\right]^{-1} \nabla_\theta \left[f'_\theta(z^*(\theta))\right] d\theta$ is the first order expansion of N_s w.r.t. θ at the point $(z^*(\theta), \theta)$ (N_s is differentiable as shown and $f'_\theta(z^*(\theta)) = 0$).

For quantity L_1, since $z^*(\theta)$ is Lipschitz continuous for any $\delta \geq 0$, we have

$$\|z^*(\theta+d\theta) - z^*(\theta)\| \leq 2\frac{L}{\mu} \|d\theta\|. \tag{4.6}$$

The Newton step has a local quadratic convergence for strongly convex $\mathcal{C}^2$ near the optimum [40]. Hence, $\|d\theta\|$ can always be chosen small enough in order to have $\|z^*(\theta+d\theta) - z^*(\theta)\|$ to be small enough to have quadratic convergence of the Newton method.

If we have quadratic convergence the following holds, for some constant K,

$$\|z^*(\theta+d\theta) - z^*(\theta) - N_s\| \leq 4K\frac{L^2}{\mu^2} \|d\theta\|^2. \tag{4.7}$$

Thus,

$$\begin{aligned}
L_1 &= \lim_{d\theta\to 0} \frac{z^*(\theta+d\theta) - z^*(\theta) - N_s}{\|d\theta\|} \\
&\leq 4K\frac{L^2}{\mu^2} \lim_{d\theta\to 0} \frac{\|d\theta\|^2}{\|d\theta\|} = 0.
\end{aligned} \tag{4.8}$$

□

4.2 Evaluation Method

Here is provided the implemented evaluation method for next presented unconstrained convex programs.

Note that other first-order and second-order methods, such as constant step gradient descent, some quasi-Newton methods, and an interior-point method, were tested but this one was the most effective in practice for us.

The method is a line search for the Newton direction, it can be found in [41] for example. It uses a backtracking scheme for the search of a point satisfying a sufficient decrease condition.

Its pseudo-code is given next, Algorithm 1. Two parameters of the method are c and ρ, those can be chosen in the $(0, 1)$ interval to obtain guaranteed convergence. The role of c is to make a trade-off between a sufficient decrease with respect to the slope at the current point and the admissible domain for the next iterate, it was fixed to $1e-4$ in the codes; ρ is used as a parameter in a geometric decrease of the next iterate distance to the current point until it satisfies the sufficient decrease condition, it is fixed to $\frac{1}{2}$ in our codes.

Data: a smooth μ-strongly convex function to optimize f with provided gradient and hessian, a precision ϵ
Result: an estimation of the optimum of the function z^* guaranteed to be within distance ϵ to the true optimum
Initialization;
$z_0 = \mathbf{0}$;
$k = 0$;
Iterations;
while $2 \|f'(x_k)\| > \mu\epsilon$ **do**
  Search direction computation;
  $f_k, f'_k, f''_k \leftarrow f(x_k)$;
  $p_k = -\left[f''(x_k)\right]^{-1} f'(x_k)$;
  Backtracking Line Search;
  while $f(x_k + \alpha p_k) \leq f(x_k) + c\alpha f'^T(x_k)p_k$ **do**
    $\alpha \leftarrow \rho\alpha$;
  end
  $k \leftarrow k + 1$;
end
Update;
$x_{k+1} \leftarrow x_k + \alpha p_k$;

Algorithm 1: optimization algorithm for smooth strongly convex functions

In practice the Hessian is not explicitly constructed, a hessian-vector product function is used instead in order to get computational advantage from potential sparsity in the problem.

An iterative method using matrix-vector products, such as the conjugate gradient method, can be used to solve the linear system.

4.3 Encoding constraints with Barriers

This is a quick description on how constraints can be handled in the case of smooth convex program. Previously it was observed that "hard" constraints posed problems at the learning optimization level due to the appearance of null non-informative gradients.

To counter this problem smooth barriers can be used instead of hard constraints. While keeping constraints satisfied they allow us to break the null gradient problem.

This is done in the next section.

4.4 Softmax

We take the last presented formulation and directly address the problem of the zero-gradient of non-active constraints.

4.4.1 Model

The model 3.12 can be rexpressed as

$$F_{A,B,b}(x) = \arg\min_{y} \max_{i \in 1...m} \{A_i y + B_i x + b_i\} + \frac{1}{2} \|y\|_2^2 , \tag{4.9}$$

with A_i, B_i, b_i the rows/elements of the respective matrix/vector with m rows/elements.

Our new model consists in replacing the max operator by a softmax operator which is differentiable, convex and hopefully will counter the inactivity of constraints problem.

The softmax operator with parameter λ applied to a vector of reals $v \in \mathbb{R}^n$ is

$$\frac{1}{\lambda} \log \left(\sum_{i=1}^{n} e^{\lambda v_i} \right). \tag{4.10}$$

Parameter λ regulates the tradeoff between smoothing the function, $\lambda \to 0$, and approximating a function close to the max operator, $\lambda \to +\infty$ (convergence to max in the limit).

This gives

$$F_{A,B,b}(x) = \arg\min_y \frac{1}{\lambda} \log\left(\sum_{i=1}^{m} e^{\lambda(A_i y + B_i x + b_i)}\right) + \frac{1}{2}\|y\|_2^2 . \tag{4.11}$$

We fix arbitrarily $\lambda = 1$ without loss of expressivity since multiplying parameters by λ yields the same optimum.

4.4.2 Gradient of Softmax model's parameters

We develop a gradient valid for this model by applying Theorem 2 for the optimum dependence with respect to the parameters A, B, b.

Let us name the objective of the program 4.11 $f(y)$ with an implicit dependence on A, B, b and x. Giving

$$f(y) = \log\left(\sum_{i=1}^{m} e^{A_i + B_i x + b_i}\right) + \|y\|_2^2 . \tag{4.12}$$

The gradient w.r.t. y is, we note exp(.) for the exponential point-wise operator, and $\sum$ without indexed variable as the sum over the components of the vector that follows it,

$$\nabla_y f(y) = \frac{\exp(Ay + Bx + b)^T A}{\sum \exp(Ay + Bx + b)} + y^T . \tag{4.13}$$

The Hessian w.r.t. y is, with $diag(v)$ for $v \in \mathbb{R}^n$ the diagonal matrix with v as diagonal,

$$\begin{aligned} \nabla_y^2 f(y) = &- \frac{\exp(Ay + Bx + b)^T A A^T \exp(Ay + Bx + b)}{\left(\sum \exp(Ay + Bx + b)\right)^2} \\ &+ \frac{A^T diag(\exp(Ay + Bx + b))A}{\sum \exp(Ay + Bx + b)} \\ &+ Id^{n \times n}. \end{aligned} \tag{4.14}$$

As in the previous cases, we pose $J(z)$ a cost function used on the optimum of the program, and we develop the associated gradient w.r.t. the parameters.

With respect to A, we note $X \cdot Y$ for the Hadamard product between two matrices or vectors X, Y (component-wise),

$$\begin{aligned}\nabla_A J(y^*) = &\frac{\nabla_z J(y^*) \left(\nabla_y^2 f(y^*)\right)^{-1} A^T \exp(Ay + Bx + b)}{\left(\sum \exp(Ay + Bx + b)\right)^2} \exp(Ay + Bx + b) y^T \\ &- \frac{\left(\left[A \left(\nabla_y^2 f(y^*)\right)^{-T} \nabla_z^T J(y^*)\right] \cdot \exp(Ay + Bx + b)\right)}{\sum \exp(Ay + Bx + b)} y^T \\ &- \frac{\exp(Ay + Bx + b) \nabla_z J(y^*) \left(\nabla_y^2 f(y^*)\right)^{-1}}{\sum \exp(Ay + Bx + b)}.\end{aligned} \tag{4.15}$$

With respect to B we find

$$\begin{aligned}\nabla_B J(y^*) = &\frac{\nabla_z J(y^*) \left(\nabla_y^2 f(y^*)\right)^{-1} A^T \exp(Ay + Bx + b)}{\left(\sum \exp(Ay + Bx + b)\right)^2} \exp(Ay + Bx + b) x^T \\ &- \frac{\left(\left[A \left(\nabla_y^2 f(y^*)\right)^{-1} \nabla_z^T J(y^*)\right] \cdot \exp(Ay + Bx + b)\right)}{\sum \exp(Ay + Bx + b)} x^T.\end{aligned} \tag{4.16}$$

And with respect to b we obtain

$$\begin{aligned}\nabla_b J(y^*) = &\frac{\nabla_z J(y^*) \left(\nabla_y^2 f(y^*)\right)^{-1} A^T \exp(Ay + Bx + b)}{\left(\sum \exp(Ay + Bx + b)\right)^2} \exp^T(Ay + Bx + b) \\ &- \frac{\left(\left[\nabla_z J(y^*) \left(\nabla_y^2 f(y^*)\right)^{-1} A^T\right] \cdot \exp^T(Ay + Bx + b)\right)}{\sum \exp(Ay + Bx + b)}.\end{aligned} \tag{4.17}$$

4.5 Sum of Non-Linear terms

Previously the max operator, or some approximation of it, was used on linear components to introduce non-linearities while making the reduction to a scalar and preserving convexity.

Another possibility is to compose with convex non-linear functions independently for each linear component (point-wise application) then to reduce the results to a scalar by summing the results (see model 4.18).

An interpretation for these types of models can be given, each term of the sum represent a preference (or equivalently reject) of some states (x, y) over others. Preferences in our interpretation is a soft version of hard constraints found in other types of logic such as propositional logic.

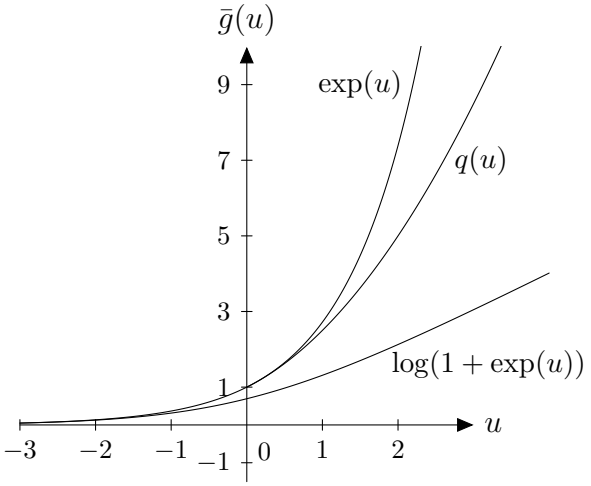

Figure 4.1: representation of the different $g(u)$ considered

4.5.1 Models

Formally the programs becomes of the type, with m the number of components,

$$F(A, B, b) = \arg\min_y \sum_{i=1}^{m} g(A_i y + B_i x + b_i) + \frac{1}{2} \|y\|_2^2 , \tag{4.18}$$

and $g : \mathbb{R} \to \mathbb{R}$ a convex function.

In this section several choices of function g are considered:

- The logistic loss function $g(u) = \log(1 + \exp u)$, which can be seen as a softmax between 0 and u.
- The exponential function.
- A function that we will call q defined as

$$q(u) = \begin{cases} \exp(u) & \text{if } u \leq 0 \\ \frac{1}{2}u^2 + u + 1 & \text{else} \end{cases} . \tag{4.19}$$

They are represented on Figure 4.1.

As we will describe the q function choice is performing best, and consequently an extension to the model 4.18 is done in an attempt to further improve the performance. We augment it with a purely quadratic term

$$F(A, B, b) = \arg\min_y \sum_{i=1}^{m} p_i g(A_i y + B_i x + b_i) + \frac{1}{2} \|Cy - d\|_2^2 + \frac{1}{2} \|y\|_2^2 , \tag{4.20}$$

where $p \in \mathbb{R}^m_+$, $C \in \mathbb{R}^{p \times n}$, and $d \in \mathbb{R}^p$ are parameters.

4.5.2 An Universal Approximation Theorem

Here we develop a theoretical understanding of the proposed model's approximation capacity.

As will be shown universal approximation holds for all functions with for domain the vertices of a hyper-cube.

Theorem 3. *Let $g : \mathbb{R} \to \mathbb{R} \in \mathcal{C}^1$ a convex function with $\lim_{x \to -\infty} g(x) = 0$. Assume g' is strictly positive, increasing at all points, and unbounded.*

Then for any function $f : \{0,1\}^p \to \mathbb{R}^n$ and any $\epsilon > 0$ there exists $A \in \mathbb{R}^{2^p \times n}$, $B \in \mathbb{R}^{2^p \times p}$, $b \in \mathbb{R}^{2^p}$ s.t.

$$\left\| f(x) - \left[y^* = \arg\min_y \sum_{i=1}^{2^p} g(A_i y + B_i x + b_i) + \frac{1}{2} \|y\|_2^2 \right] \right\| \leq \epsilon \tag{4.21}$$

holds for all $x \in \{0,1\}^n$.

Proof. The numbers $\{1 \dots, 2^p\}$ are bijectively associated to the vertices of the hyper-cube denoted by $x_i \in \{0,1\}^p \subset \mathbb{R}^p$.

With $M, T \in \mathbb{R}$ constant, let us pose $\left[B_i = M x_i^T\right]_{i \in \{1 \dots 2^p\}}$; and $b = T - pM\mathbf{1}^{2^p}$.

With these choices, $B_i x_i + b_i = M x_i^T x_i + T - pM = T$ and for $i \neq j$, $B_i x_j + b_i = M x_i^T x_j + T - pM \leq T - M$ hold.

For all $i \in \{1, \dots, 2^p\}$ there exist T_i such that the following equation in a admits a solution for all $T \geq T_i$

$$g'(a \|f(x_i)\|_2^2 + T)a = -1. \tag{4.22}$$

For proof lets first fix a, since $\lim_{a \to -\infty} g'(a \|f(x_i)\|_2^2)a = 0^-$ there exist $a < 0$ s.t. $g'(a \|f(x_i)\|_2^2)a < 0$.

Then take T_i such that $g'(a \|f(x_i)\|_2^2 + T)a \leq -1$ for $T \geq T_i$; this is always possible since g' is increasing, unbounded, and $a < 0$.

By the intermediate value theorem there exists $a^* \in (-\infty, a]$ s.t. $g'(a \|f(x_i)\|_2^2 + T)a = -1$ under $T \geq T_i$, which concludes.

Now fix $T = \max\{T_i\}_{i \in \{1 \dots 2^p\}}$, then for all $i \in \{1 \dots 2^p\}$ take a_i s.t. 4.22 is satisfied, this is well defined as just shown, and fix $A_i = a_i f(x_i)$ with x_i the vertex associated to i.

For all $i \in \{1 \dots 2^p\}$, the following holds (development below)

$$\nabla_y \left[g(A_i y + B_i x_i + b) + \frac{1}{2} \|y\|_2^2 \right] (f(x_i)) = 0. \tag{4.23}$$

It holds since the gradient at the point $f(x_i)$ is

$$g'(a \left\| f(x_i) \right\|_2^2 + T)af(x_i) + f(x_i). \tag{4.24}$$

Thus, if the following condition is valid the gradient at $f(s_i)$ is null

$$g'(a \left\| f(x_i) \right\|_2^2 + T)a + 1 = 0. \tag{4.25}$$

This is exactly 4.22, by construction it is valid.

The norm of the objective's gradient at the point $f(x_i)$ for the input x_i can be decomposed and bounded by

$$\begin{aligned}
&\left\| \sum_{j=1}^{2^p} g'(A_j f(x_i) + B_j x_i + b_j)A_j + f(x_i) \right\| \\
&\leq 0 + \sum_{j=\{1,\dots,2^p\}\setminus\{j\}} g'(A_j f(x_i) + T - M) \left\| A_j \right\| \\
&\leq (2^n - 1) \max\{\|A_j\|\}_{j\in\{1,\dots,2^p\}} \max\{g'(A_j f(x_i) + T - M)\}_{j\in\{1,\dots,2^p\}\setminus\{i\}}.
\end{aligned} \tag{4.26}$$

Since $\lim_{x\to-\infty} g'(x) = 0$, there exist M large enough s.t. the gradient above is bounded by $\frac{\epsilon}{2}$.

By strong convexity of the objective a bound on the optimal point is, with μ the strong convexity constant for our objective,

$$\|f(x_i) - y^*\| \leq \frac{2}{\mu} \left\| \sum_{j=1}^{2^p} g'(A_j f(x_i) + B_j s_j + b_j)A_j + f(x_i) \right\|. \tag{4.27}$$

Since $\mu = 1$ is valid for the objective, $\|f(x_i) - y^*\| \leq \epsilon$.

□

Corollary 1. *Models 4.18, 4.20 with $g(u) = \exp u$ or $g(u) = q(u)$ are universal approximators over the vertices of a hyper-cube.*

4.5.3 Gradients of the parameters

The gradients of the program model w.r.t. to the parameters are given.

It is done for the logistic loss choice with program 4.18 and q with program 4.20 (the exponential choice developments can also easily be derived from the development for q). Moreover, as described in the experimentation section the exponential choice is of little interest in practice.

Gradient for the Logistic Loss

As in the softmax case in the previous section (equation 4.12), let us note the objective function disregarding the dependence with the parameters

$$f(y) = \sum_{i=1}^{m} \log(1 + \exp(A_i y + B_i x + b_i)) + \frac{1}{2} \|y\|_2^2 . \tag{4.28}$$

Again we need to develop intermediate results to compute the gradient w.r.t. the parameters. Note that these intermediate results are also useful for the evaluation of the program.

Firstly, the gradient w.r.t. y is

$$\nabla_y f(y) = \left(\frac{1}{1 + \exp(Ay + Bx + b)} \right)^T A + y^T, \tag{4.29}$$

where the division and exp operator are component-wise.

Secondly, the hessian w.r.t. y

$$\begin{aligned} \nabla_y^2 f(y) = A^T diag \left(\frac{1}{\exp(-Ay - Bx - b) + \exp(Ay + Bx + B) + 2} \right) A \\ + Id^{n \times n}. \end{aligned} \tag{4.30}$$

We use again $J(z)$ as the cost function on the program output and give the gradient for it.

For A

$$\begin{aligned} \nabla_A J(y^*) = -\frac{A \left(\nabla_y^2 f(y^*) \right)^{-1} \nabla_z^T J(y^*)}{2 + \exp(Ay + Bx + b) + \exp(-Ay - Bx - b)} y^T \\ - \frac{\nabla_y f(y^*) \left(\nabla_y^2 f(y^*) \right)^{-1}}{1 + \exp(-Ay - Bx - b)}. \end{aligned} \tag{4.31}$$

For B

$$\nabla_B J(y^*) = -\frac{A \left(\nabla_y^2 f(y^*) \right)^{-1} \nabla_z^T J(y^*)}{2 + \exp(Ay + Bx + b) + \exp(-Ay - Bx - b)} x^T. \tag{4.32}$$

For b

$$\nabla_b J(y^*) = -\frac{\nabla_z^T J(y^*) \left(\nabla_y^2 f(y^*) \right)^{-1} A^T}{2 + \exp(Ay + Bx + b) + \exp(-Ay - Bx - b)}. \tag{4.33}$$

Gradient for q

We give now the gradient for program 4.20 with $g(u) = q(u)$.

The gradient is presented using the same scheme as previously and following the same notations and definitions.

The objective is

$$f(y) = p^T g(Ay + Bx + b) + \frac{1}{2} \|Cy - d\|_2^2 + \frac{1}{2} \|y\|_2^2 . \tag{4.34}$$

The binary condition in the definition of function q 4.19 forces us to define an operation to extract the relevant part of vector and matrices relatively to the activation condition ($u \leq 0$).

For a vector $v \in \mathbb{R}^p$ and a matrix $M \in \mathbb{R}^{m \times n}$ we define $v|_1$ and $M|_1$ to be a vector composed of the elements, respectively the rows for which g is in the exponential part.

We define a complementary operator $.|_2$ also defined on vectors and matrices selecting elements and respectively rows in a complementary way to the $.|_1$ operator just defined.

Also we define

$$h = Ay + Bx + b, \tag{4.35}$$

to shorten the expressions.

With the help of these notations the objective can be rewritten

$$f(y) = p|_1^T \exp(h) + p|_2^T \left(\frac{1}{2} h^2 + h + 1 \right) + \frac{1}{2} \|Cy - d\|_2^2 + \frac{1}{2} \|y\|_2^2 . \tag{4.36}$$

The gradient of the objective is then

$$\begin{aligned} \nabla_y f(y) = & \left(p|_1 \cdot \exp(h|_1)\right)^T A|_1 + \left(p|_2 \cdot (h|_2 + \mathbf{1})\right)^T A|_2 \\ & + (Cy - d)^T C + z. \end{aligned} \tag{4.37}$$

The Hessian

$$\begin{aligned} \nabla_y^2 f(y) = & A|_1^T \, diag\left(p|_1 \cdot \exp(h|_1)\right) A|_1 + A|_2^T \, diag\left(p|_2\right) A|_2 \\ & + C^T C + Id^{n \times n}. \end{aligned} \tag{4.38}$$

Finally, the gradient w.r.t. the parameters given with the help of the auxiliary function J ,are as follows

For A

$$\begin{aligned} \nabla_{A|_1} J(y^*) = & - \left(\left[A|_1 \left(\nabla_y^2 f(y^*) \right)^{-1} \nabla_z^T J(y^*) \right] \cdot p|_1 \cdot \exp(h|_1) \right) y^T \\ & - \left[p|_1 \cdot \exp(h|_1) \right] \left(\nabla_y^2 f(y^*) \right)^{-1} \nabla_z^T J(y^{*T}), \\ \nabla_{A|_2} J(y^*) = & - \left(\left[A|_2 \left(\nabla_y^2 f(y^*) \right)^{-1} \nabla_z^T J(y^*) \right] \cdot p|_2 \right) y^{*T} \\ & - \left[p|_2 \cdot (h|_2 + \mathbf{1}) \right] \left(\nabla_y^2 f(y^*) \right)^{-1} \nabla_z^T J(y^*). \end{aligned} \tag{4.39}$$

For B

$$\begin{aligned}\nabla_{B|_1} J(y^*) &= -\left(\left[A|_1 \left(\nabla_y^2 f(y^*)\right)^{-1} \nabla_z^T J(y^*)\right] \cdot p|_1 \cdot \exp(h|_1)\right) x^T, \\ \nabla_{B|_2} J(y^*) &= -\left(\left[A|_2 \left(\nabla_y^2 f(y^*)\right)^{-1} \nabla_z^T J(y^*)\right] \cdot p|_2\right) x^T.\end{aligned} \tag{4.40}$$

For b

$$\begin{aligned}\nabla_{b|_1} J(y^*) &= -\left[\nabla_z J(y^*) \left(\nabla_y^2 f(y^*)\right)^{-1} A|_1^T\right] \cdot p|_1^T \cdot \exp(h|_1^T), \\ \nabla_{b|_2} J(y^*) &= -\left[\nabla_z J(y^*) \left(\nabla_y^2 f(y^*)\right)^{-1} A|_2^T\right] \cdot p|_2^T.\end{aligned} \tag{4.41}$$

For p

$$\begin{aligned}\nabla_{p|_1} J(y^*) &= -\left[A|_1 \left(\nabla_y^2 f(y^*)\right)^{-1} \nabla_z^T J(y^*)\right] \cdot \exp(h|_1), \\ \nabla_{p|_2} J(y^*) &= -\left[A|_2 \left(\nabla_y^2 f(y^*)\right)^{-1} \nabla_z^T J(y^*)\right] \cdot (h|_2 + \mathbf{1}).\end{aligned} \tag{4.42}$$

For C

$$\begin{aligned}\nabla_C J(y^*) = &-C \left(\nabla_y^2 f(y^*)\right)^{-1} \nabla_z^T J(y^*) y^{*T} \\ &- (Cy^* - d) \nabla_z J(y^*) \left(\nabla_y^2 f(y^*)\right)^{-1}.\end{aligned} \tag{4.43}$$

For d

$$\nabla_d J(y^*) = \nabla_z J(y^*) \left(\nabla_y^2 f(y^*)\right)^{-1} C^T. \tag{4.44}$$

4.6 Experimentation

This section aims at assessing and comparing the different proposals presented earlier. First a comparison of models' capacities is done 4.6.1. From this comparison a model is selected, and further tests are done. One is to measure the effect of hidden variables 4.6.2. Finally a comparison with neural networks is done 4.6.3.

Our experimentations are done with the MNIST dataset [34]. It is a classic supervised learning problem in computer vision consisting in the identification of hand-written digits. There are 60000 28*28 pixels grayscale image of digits (0-9) in the training set and 10000 images with the same format in the test set, for each of these images an associated ground truth digit is associated.

This dataset is used for it is a problem that is known to be hard for simple supervised classifiers: linear ones achieve a 88% correct classification rate, while in comparison neural networks can achieve up to 98.4% [50].

Note that these numbers are given for methods that do not use any kind of prior on the problem, to do a fair comparison. Examples of unfair priors use include pre-processing specific to computer vision problems such as centering and dataset augmentation methods, but also specialized models such as convolutional networks.

Implementation and test details are as follows:

- The usual stochastic scheme presented in Algorithm 4 is used.
- The ADAM optimization method is used at the learning level for all models with the hyper-parameters $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 1e-8$ [29]. It is a classic method used in Deep Learning.
- The output is composed with a soft-max operator.
- The differentiable cross-entropy loss as a cost at training time, see equation 2.6.
- The q-Sum model extended has 10 quadratic terms.
- All matrix parameters are full.
- Processor used for all the experiments is an Intel Core i5-6400 2.7GHz*4, a CPU for personal computer launched in year 2015. No GPU test were performed. Since matrix multiplications are the low-level operations taking a significant share of computational time, GPU have the potential to greatly speed up the computation, as in the Deep Learning case.

4.6.1 Comparison of Smooth Strongly Convex Propositions

The different models' capacities to fit the MNIST training set are evaluated. This test does not take into account any sort of generalization performance. The goal is to assess a preliminary weaker condition to the true goal, by simply asking good performance on the training set.

Tests are done with different number terms which were observed to be the main influencer of a model capacity.

Results of the experiments are summarized in Table 4.1. The results indicates a clearly superior empirical capacity in favor of the q-Sum model with equivalent or better evaluation times.

Model	training #epochs	optimization time	accuracy on the training set
Softmax model			
with 25 terms	20	10min	84%
with 50 terms	20	30min	94%
with 100 terms	20	30min	95%
with 200 terms	20	1h15min	95%
Logistic			
with 10 terms	5	15min	90%
with 10 terms	20	1h	92%
with 10 terms	20	20min	93%
with 50 terms	20	1h20min	95%
with 100 terms	5	20min	93%
with 100 terms	20	1h30min	96%
q-Sum model			
with 10 terms	5	7min	73%
with 10 terms	20	20min	76%
with 50 terms	5	20min	96%
with 50 terms	20	1h10min	99%
with 100 terms	5	20min	97.5%
with 100 terms	20	1h15min	99.3%
with 200 terms	20	1h25min	99.3%
q-Sum model extended			
with 10 terms	5	10min	93%
with 10 terms	20	40min	95%
with 50 terms	5	25min	97%
with 50 terms	20	1h10min	99%
with 100 terms	5	20min	97.5%
with 100 terms	20	1h15min	99.5%
with 200 terms	20	1h25min	99.7%

Table 4.1: Comparison of models' capacity and optimization time. The models were trained for some number of epochs on the training part of the MNIST dataset with ADAM optimizer at the learning level. The models have all 10 variables (no hidden variables), predictions are the softmax of the optimal points. A cross-entropy loss is used, which interprets the predictions as classes' probability.

# of hidden variables	0	10	50	100
accuracy on the training set	97.3	96.6	97.4	97.6
training time	23min	34min	50min	1h

Table 4.2: Performance of the q-Sum model extended with $m = 25$ terms and 10 training epochs for different number of hidden variables.

4.6.2 Effect of Hidden Variables

Hidden variables are variables are variables jointly optimized with the variable of the output, see equation 2.4.

The effect on the model capacity of hidden variables is experimentally observed on the q-Sum extended model. Results are given Table 4.2. As can be seen the number of hidden variables does not seem to have a significant impact on the capacity of the model for this setting. Moreover, evaluation times grow.

This observation brings a more specific question: is it possible for our learning algorithm to train output and hidden variables dependencies in any useful way at all? The previous problem setting may have lacked training epochs, or the number of terms was not sufficient to profit from any beneficial effect.

To answer this question a experiment is designed where the dependence with the hidden variables must be created and used to hope any performance increase on the problem. For this the MNIST problem is kept, however, a specific structure is imposed on matrices A and B to ensure that information from the input must flow through the hidden variables to get to the output variables.

With m_1 the number of terms linking all the optimized variables; m_2 the number of terms between the hidden variables and the input; n_{in} the dimension of the input; n_o the dimension of the output variables; and n_h the dimension of the hidden variables. The structure of matrices A and B imposed is the following

$$A \in \mathbb{R}^{(m_1+m_2)\times(n_o+n_h)} = \begin{bmatrix} A_o \in \mathbb{R}^{m_1\times n_o} & A_{h1} \in \mathbb{R}^{n_1\times n_h} \\ \mathbf{0}^{m_2\times n_o} & A_{h2} \in \mathbb{R}^{m_2\times n_h} \end{bmatrix} \tag{4.45}$$

$$B \in \mathbb{R}^{(m_1+m_2)\times n_{\text{in}}} = \begin{bmatrix} \mathbf{0}^{m_1\times n_{\text{in}}} \\ B_h \in \mathbb{R}^{m_2\times n_{\text{in}}} \end{bmatrix}. \tag{4.46}$$

A test with $m_1 = 25$, $m_2 = 50$ and $n_h = 25$ for 22 epochs (3 hours) gave a training accuracy of 98%. This test indicates the possibility to learn with hidden variables.

4.6.3 Comparison with Artificial Neural Networks

Several aspects have to be presented.

The baseline optimization program that will serve for the comparison is a q-Sum extended model, model 4.20, is chosen with no hidden variables, $m = 500$ terms plus $p = 100$ quadratic terms. It is chosen on the basis of an architecture search with a validation set.

It was trained for 30 epochs (2h50min) obtaining a training accuracy of 99.8% and a generalization performance of 97.9%.

For the artificial neural networks, the already cited result of 98.4% [50] is taken as a reference. It was obtained with a 2-layers architecture having 800 hidden units.

Using the Keras framework [12], two informations were obtained: the training set accuracy when training this architecture is nearly 100% with 99.96%; also on the same hardware as for the other experiences the network takes about 20 epochs to train with about 4sec. taken by epoch.

One observation is that ANN take clearly fewer evaluations of the gradient to be train on MNIST in comparison to the optimization program. While in the same order of magnitude.

A more marked difference is the evaluation time of $70\mu s$ by image for the chosen ANN architecture versus $5600\mu s$ by image for our optimization program which is thus two order of magnitude slower.

Still, the generalization performance of our model is not very far from that of the tested ANN.

Chapter 5

Learning through Non-Convex Programs

In this part another approach to the optimization program evaluation and search direction is taken.

Previous sections presented optimization programs for which a search direction based on first-order information on the parameters-output dependence could be computed by the mean of the optimization result. The computed dependence did not need the intermediate steps taken by the optimization algorithm to compute the dependence.

This property was due to the existence of manipulable certificates of optimality, such as the gradient equal to zero in the smooth strongly convex case.

While this property is interesting since it disentangle intermediate computations from the result, it severely limit the class of optimization programs that can be used.

Here methods using other tools than certificate of optimality are developed in order to address larger class of functions.

The learning methods here are based on the computation of a gradient linking parameters to the output by backpropagation through the whole series of computation steps performed by the (lower-level) optimization algorithm. The backpropagation algorithm is of major importance in Deep Learning and many references to explain in detail it can be found, one is [43].

Developing a smooth exact minimization scheme is impossible in general for the non-convex case, the position of the optimum can be non-unique and jump with variations of parameters.

Due to this limitation heuristic non-exact optimization schemes are used for the evaluation. Two such schemes that we tested are variable cycling and

gradient descent.

Another research direction in the reinforcement learning framework is also presented.

5.1 Variable cycling

Here we wish to profit from the result we obtained for the minimizer dependence to the parameters for strongly convex functions.

The class of block-separable convex functions is considered. Given a partition of the variables, a function f is n-blocks separable convex, if there exists a partition P in n blocks of the variables such that, for all $i = 1, \ldots, n$, f restricted to the block of variables P_i at all points is convex.

This structure conveys a class of natural heuristic algorithms with it. Here we will use a cycling method, see the pseudo code Algorithm 2. It consist in cycling through the partitions in a predefined order while for each partition solving the convex problem.

Data: a n-blocks function $f(x, y)$ to minimize with respect to y, its associated partition P, x the input and a starting point y_0
Result: an heuristically obtained approximate minimizer and output of the model
Initialization;
$y \leftarrow y_0$;
Optimization of the function;
while *a predefined number of cycles are not done* **do**
  for $i \leftarrow 1$ **to** n **do**
    $y_j \in P_i \leftarrow$Optimize f over variables in P_i fixing the others to their current values;
  end
end

Algorithm 2: alternating minimization scheme for the minimization of blocks separable function

Note that this algorithm has no state dependent block selection procedure nor state dependent terminating criterion; since it is just cycling across the variables and stop after a predefined number of cycles.

Due to these properties, if the convex sub-problems are smooth and strongly convex, it is possible to get a gradient linking parameters and output using backpropagation with Theorem 2.

5.1.1 Experimentation

The function used for the experimentation is

$$f(x,y) = \sum_{i=1}^{m} q\left(y^T A_i y + b_i^T \begin{bmatrix} y \\ x \end{bmatrix} + c_i\right) + \frac{1}{2}\|y\|_2^2, \tag{5.1}$$

where q is the previously defined function 4.19, $\forall i \in 1,\dots,M \quad A_i \in \mathbb{R}^{(n+n_{\text{in}})\times(n+n_{\text{in}})}$, $b_i \in \mathbb{R}^{n+n_{\text{in}}}$, $c_i \in \mathbb{R}$, and where the diagonal elements of A_i equal to zero to guarantee uni-variate convexity.

This function is smooth and strongly convex independently in each variable at any point. Also, the function is not jointly convex for all variables.

Our main observation is that we were not able to optimize the model, the accuracy did not get better than random guesses. The experimental setting is described next.

For this experimentation MNIST is reused, see the previous experimentation section for a description. Also ADAM is used again with the same hyper-parameters, and the combination softmax-cross-entropy is reused.

Moreover, $m = 50$ was taken and 5 optimization cycles across the variables are done. These numbers were chosen as high as possible while keeping the evaluation time sufficiently low. $17000\mu s$ evaluation time by example was observed for this setting with MNIST.

Further experiments were not pursued.

5.2 Gradient descent

Here a more generic method for a larger class of functions is tested.

A basic version of the gradient method were steps proportional to the gradient are taken, see Algorithm 3.

Data: a function $f(x,y)$ to minimize with respect to y, x the input, a starting point y_0, and a scalar α regulating the size of the step taken
Result: an heuristically obtained approximate minimizer
Initialization;
$y \leftarrow y_0$;
Optimization of the function;
while *a predefined number of cycles are not done* **do**
  $y \leftarrow y - \alpha \nabla_y f(x,y)$;
end

Algorithm 3: a constant step gradient descent based approximate minimization algorithm

In a similar way to the last form presented, a backpropagation through the iterates of the method allows us to get the first-order dependency between the parameters and the output if all the operations are smooth with respect to the parameters.

5.2.1 Implementation and Experimentation

The MNIST is kept for the tests.

The chosen function f is an artificial neural network (ANN) for its ability to represent a large class of function while being differentiable (note that the sigmoid activation function is used, see below).

More precisely, evaluation time was observed to be a clear limitation of the method, thus the choice is made to use an one hidden layer ANN with only 20 hidden nodes. Following the same logic no hidden variable were used.

To ensure the existence of an optimum a quadratic term is added.

The final function is, with $\sigma(u) = \frac{1}{1+e^{-u}}$ the sigmoid function applied point-wisely,

$$f(x, y) = W_2\sigma\left(W_1^y y + W_1^x x + b\right) + \frac{10^{-4}}{2}\|y\|_2^2, \tag{5.2}$$

with $W_1^x \in \mathbb{R}^{20\times 28^2}$, $W_1^y \in \mathbb{R}^{20\times 10}$, $b \in \mathbb{R}^{20}$, $W_2 \in \mathbb{R}^{1\times 20}$.

For the implementation the Julia package Flux.jl [26] is used, Flux.jl helps by being able to automatize the computation of gradients. Since the gradient of the parameters is found by passing second-order dependence with y and f, the package is highly recommended to lower the coding and gradient computation burden.

The evaluation time of the gradient for one sample is $90000\mu s$ in this setting. This time is does not allows us to optimize on the full MNIST dataset, one epoch requiring 1h30min.

An attempt to fit the model on a subset of 1000 training samples with ADAM reveals that the model is hard to train. At 30 epochs a 96% training accuracy was attained, this accuracy is low compared to other approaches even on the full training set and with less computation time. Moreover, the accuracy started to decrease in subsequent training epochs.

This learning algorithm is thus not practicable.

5.3 Reinforcement learning: another research direction

The ideas to learn non-convex programs were based on usual non-convex algorithms associated with backpropagation. Another possibility is to fit the tools of reinforcement learning to our problem [56].

In reinforcement learning the problem of optimizing a function can be posed as taking a sequence of actions decreasing a value function, which can be simply considered as the objective function. Moreover, a reward corresponding to the fitness of the current state with the expected output can be posed at the end of a sequence of optimization actions.

Once these two links are made a reinforcement learning algorithm can be transformed into a supervised learning algorithm. The main gain in this change with respect to our approach is more freedom, for the actions must no more be differentiable to apply reinforcement learning algorithms.

Chapter 6

Optimization Programs as Components

Practical learning of functions expressed as convex programs from input-output pairs was developed in chapter 4. Our learning algorithm is gradient-based, this property allows us to extend the results obtained using the back-propagation algorithm.

More precisely, the fact that the learning algorithm is gradient-based allows to extend it to composition of differentiable and learnable by gradient descent models. This is the same idea as in Deep Learning with a composition of simple models fitted by gradient descent using gradient propagation across models.

The models developed can thus be inserted inside known Deep Learning architectures without re-thinking the optimization.

This last decade Deep Learning research has produced new architectures that less and less resemble the first designed networks, up to a point where Differentiable Programming was proposed as a new name for the field.

Some of the proposed differentiable programs inserted some optimization program as component. A first example is replacing current simple pooling procedures used in convolutional neural networks by a more nuanced voter scheme [25]. In that papers the authors pose it as a clustering optimization problem solved by an iterative expectation maximization algorithm with closed form solutions, see [8] for an explanation of the expectation maximization algorithm. The learning algorithm is gradient-based and uses backpropagation through the steps.

The second example aims at obtaining of useful representations of images in an unsupervised way [21]. Each image has unknown descriptors, and the link from the descriptors to the images is plastic and can be learned. Moreover, the unknown descriptors of an image can be retrieved using (again)

an expectation maximization algorithm. However, in this case the iterative steps does not have analytical solutions and the authors decide to use gradient descent instead in the maximization step. The learning algorithm optimizes the link from the descriptors to the images by backpropagating through the expectation-maximization/gradient descent algorithm.

The learning algorithms associated with these proposals are conceptually close to the ideas presented in the previous chapter, but applied to other classes of optimization problems. The link is that the algorithm backpropagates through a sequence of optimization steps applied on a non-convex problem. A contribution of this work for these type of problems is to show that if the problem can decomposed in an iterative resolution of convex programs, then analytical solutions for the convex problems are not necessary for backpropagation.

Next, we present a simple experimental setup to test the composition of optimization program models.

6.1 Experimentation

Here is reported a simple experiment concretizing under a simple form the possible extension described above. It is chosen to compose three optimization program. A model to fit (again) the MNIST dataset is learned out of it using backpropagation.

The exact architecture is a composition of q-Sum extended model with no hidden variables and outputs of size 20, 10, 10 respectively and all three with $m = 50$ terms and 10 quadratic terms.

The rest of the experimentation setting is the same as in the previous experimentation on this model.

The result after 25 epochs of training (3h50min), is an accuracy of 99.4% on the training set and 97.3% on the test set (it was 97.9% for our single q-Sum extended model compared with ANN).

Note that the parameters of the architecture were not optimized. This result shows that it is possible to learn through composition of optimization programs.

Chapter 7

Higher-Order Structures Learning

In this chapter we explore ways to learn higher-order patterns in the optimization framework. To clarify, higher-order in the logical sense is taken not the higher-order polynomial sense.

Logical higher-order patterns appears when the description of a theory/-model is factorized or can be factorized in smaller parts. It is one of the main difference between predicate logic (first-order logic) and propositional logic (zeroth-order logic). Predicate logic can explicitly describe higher-order structures in a theory. For a basic introduction to these theories a reference is [44].

The following section explores the problem of learning theories in a logical language expressive enough to capture higher-order patterns.

7.1 Learning Predicates

Here we look at the problem of constructing an algorithm that can learn first-order logic theories and make the observation that using learning through optimization programs could be part of a solution.

Pertinence of the problem Developing such an algorithm allows to capture higher-order patterns in data. Benefiting from these patterns is important in the same way that first-order logic is important to describe structure that propositional logic cannot. Not using these patterns leads to more complex and more difficult models than need to be.

Predicates The sub-problem on which we focus is the learning of a predicate. For a given theory to learn, in our setting, a predicate will be a function from tuples of objects belonging to the theory to a real number to which a meaning of truth can be associated. We will assume that the set of objects belonging to the theory is finite.

For example let's take the classic family theory. In family theory, the objects are persons of the family, and the predicate encode relationships between the persons. An example of predicate could be $P(x, y, z) \geq$ some real number iff x is the child of y and z. Other example of predicates could relate to the truth of the relationships: grandfather, uncle, son, daughter, is a male, is a female, belong to the same family, ...

In the example just taken the relationship between truth and the real number returned by the predicate function can seem a bit artificial, why not just use a binary system to represent predicate's truth? This remark make sense in this case because the predicate has a clear meaning and value from an human point of view. Thus it does not point a clear reason to restrict the expressivity from real number to binary value for learned predicates.

Note that if one wants to learn a predicate, the predicate does not necessarily have to possess a predefined meaning, the concept it has to represent can be left to the learning algorithm, as for hidden neurons in Deep Learning.

Appearing difficulty Now suppose one wants to learn and use a predicate function inside a more complete system with logic inference. At inference time the elements of the domain of the predicate have to be considered. This poses a serious challenge since the space of tuples grows exponentially with the length.

One approach One possible approach would be to pose any question at inference time on a predicate domain as an optimization problem with some structure. This approach could then profit from the capacity of optimization algorithm to find an answer with limited computations using potential struture in the problem.

One way to do this is to use as a variable a matrix representing a tuple. For any tuple of t object a real matrix T of size $p \times n$ (n is the number of objects in the theory) can be constructed with one-hot encoding, for each ith row all entry are zero except the one corresponding to the object in position i whose value is one.

The selection of the objects can then be written as a matrix multiplication

$T \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}$, if the left multiplication by a real and a sum operator are defined. A natural choice would be to embed the objects in real vectors to have this property reducing the operation to a matrix-matrix operation. With a space of embedding of dimension q, let's note $X = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} \in \mathbb{R}^{n \times q}$.

With this tuple and object representation, an inference with only one predicate could be an optimization program of the form

$$\underset{T \in \text{objects tuple space}}{\arg\min} \quad L_P(TX), \tag{7.1}$$

where L_P is a learned loss for predicate P, X is the learned embedding. The learning problem can profit from the development done in the previous chapter, notice that post multiplying by X is a linear operator and thus preserve the potential convexity of function L_P.

A satisfying inference procedure was not designed. Nonetheless, here is an example that introduces a way of constructing one and highlight the importance of the optimization framework in this problem.

Be the rule $\mathit{Father}(x, y)$ and $\mathit{Father}(y, z) \rightarrow \mathit{GrandFather}(x, z)$[1].

A possibility is to pose the inference problem as

$$\begin{aligned} \underset{T_1, T_2 \in \text{objects 2-tuple space}}{\arg\min} \quad & L_{\mathit{Father}}(T_1 X) + L_{\mathit{Father}}(T_2 X) \\ \text{s.t.} \quad & T_1(2) = T_2(1). \end{aligned} \tag{7.2}$$

Here the capacity to encode constraints in the optimization framework is essential.

[1]This rule is not formally defined in our used logic, it is chosen to stay informal.

Chapter 8

Conclusion

A new class of parametrized functions, based on the use of an inner optimization model, has been introduced and tested on Machine Learning tasks. A universal approximation theorem for binary vectors in input was shown to hold.

We report for this Machine Learning algorithm a 97.9% classification accuracy on the classic MNIST dataset. While practicable, the algorithm does not reach the computational performance of other currently used machine learning tools.

Experimentation also taught us that the learning algorithm could make use of potential hidden variables in the optimization program.

An interesting property is that the model developed has the flexibility to be taken as a module inside larger differentiable Deep Learning architectures.

A pragmatic value to these developments has to be found in specific problems. Potential applications were shown. However, the applications presented still require work to integrate the obtained results and validate their usefulness.

Non-convex approaches were tested without any success to report, but an encouraging research direction is given in a reinforcement learning formulation.

Acknowledgement

I wish to thank Prof. François Glineur for his endorsement and the support he has given to this work through the time he has given in rich comments and discussions.

Appendix A

Pseudo-codes

A.1 Supervised learning

Data: $xData$, $yData$
Result: a function making predictions from the $xData$ space to the $yData$ space
Initialization of the function parameters;
$\theta \leftarrow$ `RandomInitialization(`$xData, yData$`)`;
Initialization of the optimizer;
$\mathbb{O} \leftarrow$ `InitializeOptimizer()`;
Optimization of the function parameters;
for $e \leftarrow 1$ **to** $nEpochs$ **do**
  for $i \leftarrow 1$ **to** $nSamples$ **do**
    Evaluate the function on the sample;
    $z^* \leftarrow \arg\min_z$`(`$f(z, xData_i, \theta)$`)`;
    Evaluate gradient;
    $g \leftarrow$ `EvaluateMetaGradient(`$Lossfunction, \theta, z^*, yData_i$`)`;
    Update Optimizer;
    $\mathbb{O} \leftarrow$ `Update`$\mathbb{O}$`(`$\mathbb{O}, g$`)`;
    Update θ;
    $\theta \leftarrow \mathbb{O}$`(`$\theta, g$`)`;
  end
end
return $P(x) = \arg\min_z$`(`$f(z, x, \theta)$`)`;

Algorithm 4: supervised learning with a stochastic algorithm for bilevel optimization

Bibliography

[1] Amouzegar, M. A. and Moshirvaziri, K. (1999). Determining optimal pollution control policies: An application of bilevel programming. *European Journal of Operational Research*, 119(1):100–120.

[2] Bard, J. F. and Falk, J. E. (1982). An explicit solution to the multi-level programming problem. *Computers & Operations Research*, 9(1):77–100.

[3] Bard, J. F. and Moore, J. T. (1990). A branch and bound algorithm for the bilevel programming problem. *SIAM Journal on Scientific and Statistical Computing*, 11(2):281–292.

[4] Bard, J. F. and Moore, J. T. (1992). An algorithm for the discrete bilevel programming problem. *Naval Research Logistics (NRL)*, 39(3):419–435.

[5] Ben-Ayed, O., Blair, C. E., Boyce, D. E., and LeBlanc, L. J. (1992). Construction of a real-world bilevel linear programming model of the highway network design problem. *Annals of Operations Research*, 34(1):219–254.

[6] Bennett, K. P., Hu, J., Ji, X., Kunapuli, G., and Pang, J.-S. (2006). Model selection via bilevel optimization. In *The 2006 IEEE International Joint Conference on Neural Network Proceedings*, pages 1922–1929. IEEE.

[7] Bennett, K. P., Kunapuli, G., Hu, J., and Pang, J.-S. (2008). Bilevel optimization and machine learning. In *IEEE World Congress on Computational Intelligence*, pages 25–47. Springer.

[8] Bishop, C. M. (2006). *Pattern recognition and machine learning*. springer.

[9] Bracken, J. and McGill, J. T. (1974). Defense applications of mathematical programs with optimization problems in the constraints. *Operations Research*, 22(5):1086–1096.

[10] Brotcorne, L., Labbé, M., Marcotte, P., and Savard, G. (2001). A bilevel model for toll optimization on a multicommodity transportation network. *Transportation science*, 35(4):345–358.

[11] Brown, G., Carlyle, M., Diehl, D., Kline, J., and Wood, K. (2005). A two-sided optimization for theater ballistic missile defense. *Operations research*, 53(5):745–763.

[12] Chollet, F. et al. (2015). Keras. `https://keras.io`.

[13] Colson, B., Marcotte, P., and Savard, G. (2005). A trust-region method for nonlinear bilevel programming: algorithm and computational experience. *Computational Optimization and Applications*, 30(3):211–227.

[14] Colson, B., Marcotte, P., and Savard, G. (2007). An overview of bilevel optimization. *Annals of operations research*, 153(1):235–256.

[15] Dempe, S. (2002). *Foundations of bilevel programming*. Springer Science & Business Media.

[16] Deng, X. (1998). Complexity issues in bilevel linear programming. In *Multilevel optimization: Algorithms and applications*, pages 149–164. Springer.

[17] Edmunds, T. A. and Bard, J. F. (1991). Algorithms for nonlinear bilevel mathematical programs. *IEEE transactions on Systems, Man, and Cybernetics*, 21(1):83–89.

[18] Franceschi, L., Frasconi, P., Salzo, S., and Pontil, M. (2018). Bilevel programming for hyperparameter optimization and meta-learning. *arXiv preprint arXiv:1806.04910*.

[19] Golub, G. H. and Van Loan, C. F. (2012). *Matrix computations*, volume 3. JHU press.

[20] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep learning*. MIT press.

[21] Greff, K., van Steenkiste, S., and Schmidhuber, J. (2017). Neural expectation maximization. In *Advances in Neural Information Processing Systems*, pages 6691–6701.

[22] Hansen, P., Jaumard, B., and Savard, G. (1992). New branch-and-bound rules for linear bilevel programming. *SIAM Journal on scientific and Statistical Computing*, 13(5):1194–1217.

[23] Hejazi, S. R., Memariani, A., Jahanshahloo, G., and Sepehri, M. M. (2002). Linear bilevel programming solution by genetic algorithm. *Computers & Operations Research*, 29(13):1913–1925.

[24] Herskovits, J., Leontiev, A., Dias, G., and Santos, G. (2000). Contact shape optimization: a bilevel programming approach. *Structural and multidisciplinary optimization*, 20(3):214–221.

[25] Hinton, G. E., Sabour, S., and Frosst, N. (2018). Matrix capsules with em routing.

[26] Innes, M. (2018). Flux: Elegant machine learning with julia. *Journal of Open Source Software*.

[27] Jeroslow, R. G. (1985). The polynomial hierarchy and a simple model for competitive analysis. *Mathematical programming*, 32(2):146–164.

[28] Johnson, M., Aghasadeghi, N., and Bretl, T. (2013). Inverse optimal control for deterministic continuous-time nonlinear systems. In *52nd IEEE Conference on Decision and Control*, pages 2906–2913. IEEE.

[29] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[30] Kolstad, C. D. and Lasdon, L. S. (1990). Derivative evaluation and computational experience with large bilevel mathematical programs. *Journal of optimization theory and applications*, 65(3):485–499.

[31] Laffont, J.-J. and Martimort, D. (2009). *The theory of incentives: the principal-agent model*. Princeton university press.

[32] Lancia, G. and Serafini, P. (2014). Deriving compact extended formulations via lp-based separation techniques. *4OR*, 12(3):201–234.

[33] LeBlanc, L. J. and Boyce, D. E. (1986). A bilevel programming algorithm for exact solution of the network design problem with user-optimal flows. *Transportation Research Part B: Methodological*, 20(3):259–265.

[34] LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., et al. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

[35] Li, X., Tian, P., and Min, X. (2006). A hierarchical particle swarm optimization for solving bilevel programming problems. In *International Conference on Artificial Intelligence and Soft Computing*, pages 1169–1178. Springer.

[36] Liang, J. Z. and Miikkulainen, R. (2015). Evolutionary bilevel optimization for complex control tasks. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pages 871–878. ACM.

[37] Mathieu, R., Pittard, L., and Anandalingam, G. (1994). Genetic algorithm based approach to bi-level linear programming. *RAIRO-Operations Research-Recherche Opérationnelle*, 28(1):1–21.

[38] Mitchell, T. M. (1980). *The need for biases in learning generalizations*. Department of Computer Science, Laboratory for Computer Science Research ....

[39] Moore, J. T. and Bard, J. F. (1990). The mixed integer linear bilevel programming problem. *Operations research*, 38(5):911–921.

[40] Nesterov, Y. (2004). *Introductory lectures on convex optimization: A basic course*, volume 87. Springer Science & Business Media.

[41] Nocedal, J. and Wright, S. (2006). *Numerical optimization*. Springer Science & Business Media.

[42] Petersen, K. B., Pedersen, M. S., et al. (2008). The matrix cookbook. *Technical University of Denmark*, 7(15):510.

[43] Rumelhart, D. E., Durbin, R., Golden, R., and Chauvin, Y. (1995). Backpropagation: The basic theory. *Backpropagation: Theory, architectures and applications*, pages 1–34.

[44] Russell, S. J. and Norvig, P. (2016). *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,.

[45] Savard, G. and Gauvin, J. (1994). The steepest descent direction for the nonlinear bilevel programming problem. *Operations Research Letters*, 15(5):265–272.

[46] Seider, W. D. and White III, C. W. (1985). Chemical reaction equilibrium analysis: Theory and algorithms by william r. smith and ronald w. missen, 364 pp., john wiley, 1983, 42.95. *AIChE Journal*, 31(1):176–176.

[47] Sherali, H. D., Soyster, A. L., and Murphy, F. H. (1983). Stackelberg-nash-cournot equilibria: characterizations and computations. *Operations Research*, 31(2):253–276.

[48] Shimizu, K. and Aiyoshi, E. (1981). A new computational method for stackelberg and min-max problems by use of a penalty method. *IEEE Transactions on Automatic Control*, 26(2):460–466.

[49] Shimizu, K., Ishizuka, Y., and Bard, J. (1997). Nondifferentiable and two-level mathematical programming.

[50] Simard, P. Y., Steinkraus, D., Platt, J. C., et al. (2003). Best practices for convolutional neural networks applied to visual document analysis. In *Icdar*, volume 3.

[51] Sinha, A., Malo, P., and Deb, K. (2013a). Efficient evolutionary algorithm for single-objective bilevel optimization. *arXiv preprint arXiv:1303.3901*.

[52] Sinha, A., Malo, P., and Deb, K. (2018). A review on bilevel optimization: from classical to evolutionary approaches and applications. *IEEE Transactions on Evolutionary Computation*, 22(2):276–295.

[53] Sinha, A., Malo, P., Frantsev, A., and Deb, K. (2013b). Multi-objective stackelberg game between a regulating authority and a mining company: A case study in environmental economics. In *2013 Ieee Congress on Evolutionary Computation*, pages 478–485. IEEE.

[54] Sinha, A., Malo, P., Xu, P., and Deb, K. (2014). A bilevel optimization approach to automated parameter tuning. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, pages 847–854. ACM.

[55] Suryan, V., Sinha, A., Malo, P., and Deb, K. (2016). Handling inverse optimal control problems using evolutionary bilevel optimization. In *2016 IEEE Congress on Evolutionary Computation (CEC)*, pages 1893–1900. IEEE.

[56] Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

[57] Vicente, L., Savard, G., and Judice, J. (1996). Discrete linear bilevel programming problem. *Journal of optimization theory and applications*, 89(3):597–614.

[58] Ye, J. J. and Zhu, D. (2010). New necessary optimality conditions for bilevel programs by combining the mpec and value function approaches. *SIAM Journal on Optimization*, 20(4):1885–1905.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**