**UNIVERSIDADE FEDERAL DE MINAS GERAIS**
**Instituto de Ciências Exatas**
**Programa de Pós-Graduação em Ciência da Computação**

Francisco Galuppo Azevedo

# Programação Dinâmica Neural Estocástica

Belo Horizonte
2023

Francisco Galuppo Azevedo

# Programação Dinâmica Neural Estocástica

**Versão Final**

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Minas Gerais, como requisito parcial à obtenção do título de Mestre em Ciência da Computação.

Orientador: Adriano Alonso Veloso

Belo Horizonte
2023

Francisco Galuppo Azevedo

# Stochastic Neural Dynamic Programming

**Final Version**

Thesis presented to the Graduate Program in Computer Science of the Federal University of Minas Gerais in partial fulfillment of the requirements for the degree of Master in Computer Science.

Advisor: Adriano Alonso Veloso

Belo Horizonte
2023

Azevedo, Francisco Galuppo.

A994s Stochastic neural dynamic programming [recurso eletrônico] / Francisco Galuppo Azevedo - 2023.
1 recurso online (124 f. il., color.) : pdf.

Orientador: Adriano Alonso Veloso.

Dissertação (Mestrado) - Universidade Federal de Minas Gerais, Instituto de Ciências Exatas, Departamento de Ciência da Computação.
Referências: f. 115-124

1. Computação – Teses. 2. Aprendizado profundo – Teses. 3. Otimização matemática -Teses. 4. Programação dinâmica – Teses. I. Veloso, Adriano Alonso Veloso. II. Universidade Federal de Minas Gerais, Instituto de Ciências Exatas, Departamento de Ciência da Computação. III. Título.

CDU 519.6*82(043)

Ficha catalográfica elaborada pela bibliotecária Irénquer Vismeg Lucas Cruz
CRB 6/819 - Universidade Federal de Minas Gerais - ICEx


UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

# FOLHA DE APROVAÇÃO

Programação Dinâmica Neural Estocástica

## FRANCISCO GALUPPO AZEVEDO

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

Adriano Alonso Veloso

PROF. ADRIANO ALONSO VELOSO - Orientador
Departamento de Ciência da Computação - UFMG

Wagner Meira Jr.

PROF. WAGNER MEIRA JÚNIOR
Departamento de Ciência da Computação - UFMG

Ram Rajagopal

DR. RAM RAJAGOPAL
Civil and Environmental Engineering - Stanford University

Belo Horizonte, 12 de setembro de 2023.

"*Eu vos digo: é preciso ter ainda caos dentro de si, para poder dar à luz uma estrela dançante. Eu vos digo: tendes ainda caos dentro de vós.*"
(Friedrich Nietzsche, Assim Falou Zaratustra)

# Resumo

Problemas de Otimização Estocástica de Múltiplos Estágios (MSSO) são comuns em cenários de tomada de decisão que envolvem incertezas sequenciais, abrangendo várias áreas, como logística de transporte, configuração de infraestrutura e agricultura. Este trabalho apresenta duas contribuições importantes: um novo algoritmo para resolver problemas MSSO, sejam eles convexos ou não, e um conjunto de metodologias para analisar o comportamento desse algoritmo e de outros em relação à tomada de decisões.

O algoritmo proposto, Programação Dinâmica Neural Estocástica, aproveita as capacidades preditivas das redes neurais para lidar com problemas MSSO. Superando desafios relacionados à modelagem das funções de custo futuras como redes neurais e ao treinamento de seus pesos, o algoritmo emprega um modelo de programação inteira e uma nova função de perda. A avaliação experimental em diferentes problemas demonstra a capacidade do algoritmo de treinar efetivamente os pesos da rede. No entanto, a complexidade computacional surge como uma limitação, especialmente em problemas com muitas variáveis de decisão e estágios, devido à necessidade de resolver um grande número de subproblemas de Programação Linear Mista (MILP) ou Programação Quadrática Mista (MIQP).

Uma vantagem significativa do uso de redes neurais reside na capacidade de otimizar além de valores médios, considerando observações, informações latentes e tendências temporais, libertando-se das restrições de independência probabilística entre os estágios.

Além de resolver os problemas propostos, o texto aborda a importância de compreender a lógica por trás das decisões e os riscos associados a aplicações de longo prazo no mundo real. Ele apresenta um conjunto de metodologias que utilizam o SHAP (SHapley Additive exPlanations) e a curvatura das superfícies. Essas técnicas fornecem informações sobre o impacto dos reservatórios e outras variáveis nos custos futuros e suas interações. Além disso, o uso de métricas de curvatura permite a quantificação do comportamento local na função de custo futura, facilitando a análise ao migrar a política aprendida para cenários do mundo real.

**Palavras-chave:** aprendizado profundo; otimização; despacho hidrotérmico; fluxo de potência.

# Abstract

Multi-Stage Stochastic Optimization (MSSO) problems are prevalent in decision-making scenarios involving sequential uncertainty, spanning various domains such as transport logistics, infrastructure configuration, and agriculture. This work presents two key contributions: a novel algorithm for solving MSSO problems, whether convex or not, and a set of methodologies to analyze the behavior of this algorithm and others concerning decision-making.

The proposed algorithm, Stochastic Neural Dynamic Programming, leverages the predictive capabilities of neural networks to address MSSO problems. Overcoming challenges related to modeling future cost functions as neural networks and training their weights, the algorithm employs an integer programming model and a novel loss function. Experimental evaluation on different problems demonstrates the algorithm's ability to train network weights effectively. However, computational complexity emerges as a limitation, particularly in problems with numerous decision variables and stages, due to the need to solve a substantial number of MILP (Mixed-Integer Linear Programming) or MIQP (Mixed-Integer Quadratic Programming) subproblems.

A notable strength of using neural networks lies in their capacity to optimize beyond average values by considering observations, latent information, and temporal trends, breaking free from the constraints of probabilistic independence between stages.

In addition to solving the proposed problems, the text addresses the importance of understanding the rationale behind decisions and associated risks for long-term real-world applications. It introduces a set of methodologies utilizing SHAP (SHapley Additive exPlanations) and curvature of surfaces. Furthermore, the use of curvature metrics enables the quantification of local behavior in the future cost function, facilitating analysis when migrating the learned policy to real-world scenarios.

**Keywords:** deep learning; optimization; hydrothermal dispatch; power flow.

# List of Figures

# List of Tables

# List of Algorithms



*

# Contents

# Chapter 1

# Introduction

Long-term decision making is a difficult task. If on the one hand you know the immediate consequences of a decision, on the other hand the impacts of future events are uncertain. A situation like this is the problems of resource management, in which we have a stock of a product, a water reservoir, a financial portfolio; situations in which demand, rainfall and future prices are uncertain. This work seeks to advance over a subset of problems of this type, in which the decisions of each stage are limited by known restrictions, the objective function is also known and we have realizations, historical or simulated, of the variables with uncertain. We will call this type of problem of Multi-Stage Stochastic Optimization (MSSO).

In this first chapter we have a general overview of problems of this type, a discussion of the modeled problems at work, the proposed research objectives and the description of the work structure.

## 1.1 Multi-Stage Stochastic Optimization

A stochastic optimization problem is an optimization program in which some or all the model parameters are uncertain, but past realizations (scenarios) are known. A multi-stage stochastic optimization problem (MSSO) is when we have a stochastic optimization in which decisions are made sequentially, as more information becomes available. For the purposes of this work, it is assumed that a MSSO is composed of three types of variables:

(a) **State**: variables that track a system property over time;

(b) **Control**: the actions taken at each stage that modify the state variables;

(c) **Random**: random variables observed at the beginning of a stage, before the control.

Algorithms to solve MSSOs, such as dynamic programming, usually suffer from the need to discretize the decision space. With this, the number of combinations of decision

variables increases exponentially in the size of the state and in the number of stages [86, 23]. An algorithm that has exceeded this limitation was the so-called Stochastic Dual Dynamic Programming (SDDP) [86], which has become the state-of-the-art for convex MSSO [102] and is used in various hydrothermal systems around the world [31, 70]. Its solution was a decomposition of stages with the purpose of approximating to each a future cost function, convex and piciewise linear, over the values of the state. Thus, it is no longer necessary to discretize and list the decision variables, and it makes it possible to find solutions for large-scale systems. The algorithm proposed here in this work is an extension of the SDDP, in which it is sought to generalize it for non-convex functions.

Many real decision-taking situations involve sequential uncertainty, and for this can be modeled as MSSOs. The transport logistics area has works such as the allocation of a fleet of rental cars [47], the distribution of the collection and blood storage for hospitals and clinics [113], the management of bus *stop-skipping* routes [54]. There are also applications for the configuration of infrastructure such as the expansion of the natural gas infrastructure [48] and the allocation of resources for the relief of natural disasters [53]. Recently, some work in the agricultural sector has adopted the modeling of multi-stage stochastic optimization, for example for irrigation control [64], the management of a dairy farm [34] and the labor management in the winery harvest period [7].

Although the proposed algorithm is general for the resolution of all these applications, the work focuses on two MSSOs connected to the energy sector: the hydrothermal dispatch and the optimal power flow, discussed below.

## 1.2 Hydrothermal Dispatch

Hydrothermal systems are electrical energy generation systems composed of hydroelectric and thermal plants, which together must generate enough energy to meet a given demand. The two types of plants differ in various points. The cost per generation of an energy unit is considered negligible for hydroelectric plants and constant for thermal plants, whereas the costs of these latter can vary between them. Thus, there is a direct preference for the use of energy of hydroelectric plants, free of cost. These, however, have their generation limited by the volume of water that is found in their reservoirs, and can not always meet the system demand. The problem with the dispatch of hydrothermal systems is the management of the use of this water at each stage (e.g., every month) in order to minimize the total cost of the thermeletrics over a long operation period (e.g., two years).

Different factors make it difficult to make this decision in addition to this time

coupling, in particular the spatial coupling and the variability of rain. In the first, the spatial coupling, we have to coordinate the decisions of all hydroelectric in a joint manner, since the water outflowing a reservoir feeds other lower reservoirs. There is also a stochastic component of the water inflow coming from the rain, in which we don't know how much more water we will have in the future. One can opt to spare water at the current stage in order to take precautions in case the next stage is drier, but it can also entail an additional mandatory cost if the next stage is heavy rain. It is thus a MSSO [13].

The problem of hydrothermal systems has been formulated as a problem of optimization for the first time in [21]. Since then, it has been extensively studied, resulting in stochastic [85, 20, 100] and deterministic [101, 72] formulations, which vary depending on whether the random nature of rainwater flows is considered or not.

By minimizing a coherent risk measure [6], more commonly the average, SDDP seeks to find, for each stage, a function that well represents the future cost behavior for the different historical scenarios, even if the rain trajectory are different. Under the context of climate change [84], future rainfall becomes quite difficult to estimate by historical data due to different effects, such as the reduction in volume and river flow due to precipitation changes, the occurrence of extreme effects and the shift of spatial patterns. In Brazil, for example, the frequency and intensity of droughts have grown in the last decade, culminating in the worst drought recorded, in 2021, which peaked the cost of energy due to the use of thermeletrics. Therefore, it is of interest not only to learn a future cost function for each stage but also to make corrections of this conditioned function in other information beyond the historical, such as recent rainfall and other climate variables.

## 1.3 Optimal Power Flow

If on the one hand the incorporation of renewable energy sources offer a decarbonization of the economy, it also causes the addition of uncertainty to the electric grid [1, 18, 16], which should adapt. A approach to dealing with this operational challenge is the use of batteries as distributed energy reservoirs [9]. With storage technologies it is possible to delay the need for greater generation, improve the distribution of demand, act as energy reserve and correct frequency and power [96].

The problem of Optimal Power Flow (OPF) [17] for alternate current circuits (AC) in its original definition relates to finding the power of the generators and the best flow distribution on the grid to meet the various points of demand, with the aim of minimizing

a cost function, which can be the cost of the generators, the losses for resistance, among others. With the incorporation of the batteries, the system becomes a MSSO, in which you have to decide on the charge and discharge of the batteries. The problem has multiple restrictions, such as limits for power transfer and thermal losses, as well as physical restrictions of the alternate current [42, 43].

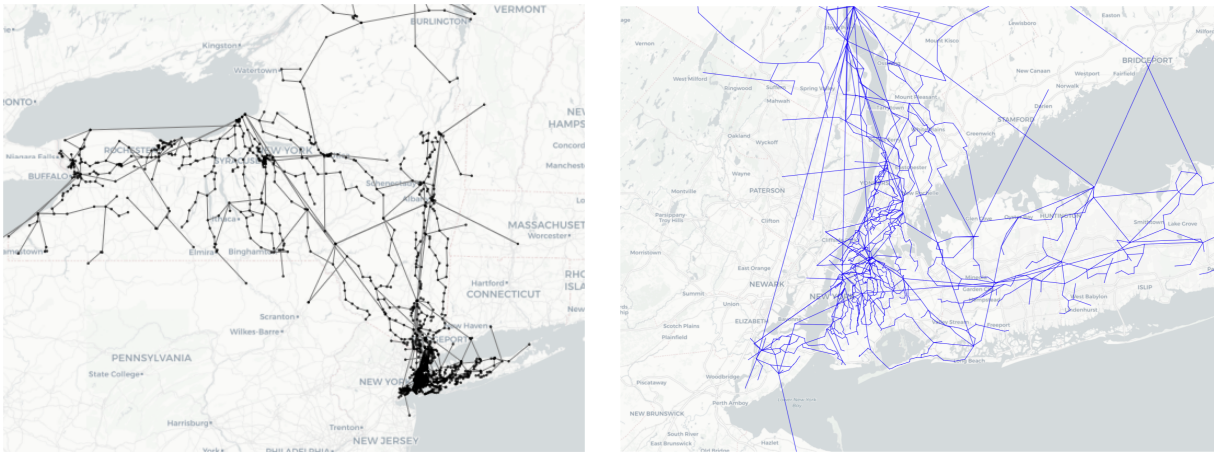


Figure 1.1: Topology of the New York system, in the U.S., consisting of 1814 buses and more than 500 generators. It is not considered a big system. Figure by [12].

## 1.4 Objectives

This work mainly aims to develop a new algorithm for multi-stage stochastic optimization problems, whether it is convex or not, with stagewise dependency between random variables or not, through the use of neural networks. We will call this algorithm of Stochastic Neural Dynamic Programming (SNDP). In particular, it aims to obtain such algorithm as an extension of the SDDP algorithm. It is therefore vital to substantiate the reader with the necessary knowledge about this latter so that the next step is as easy as possible.

As will be discussed in more detail in section 3.2, the SDDP algorithm works by approaching the future cost of a decision through a convex, easy-to-optimize function. The goal of our algorithm is to approximate that future cost as a neural network, thus being able to model a greater range of future cost functions, and allowing the use of other entries, such as data correlated with the uncertain variables or time trends.

One interest of the researchers was also to better understand the future cost functions learned by the models. This can be achieved in two ways: seeking to measure the

impact of a state on the future cost function and studying the mathematical properties of the functions.

More specifically, the objectives of this work are to:

1) Propose a neural algorithm that learns the future cost functions like SDDP;

2) Apply the algorithm to different optimization problems, synthetic or real;

3) Check if the proposed model deals with extreme scenarios properly;

4) Study the reasons behind the actions of the algorithm with explainability techniques;

5) Study the properties of the future cost function through the function curvature.

## 1.5 Work Structure

The rest of the work is divided into 7 chapters. The idea is to bring a panorama of the related work and an introduction to the necessary theoretical concepts, then present the generated content, the experiments that testify its effectiveness and finally the conclusion of the work. The next two chapters, 2 e 3, act in this first part, in which the first presents the classic work of the area of optimization as well as new neural approaches to problems of the type; and the second brings the necessary coverage for what will be presented later.

Chapter 4 presents the formal definition of the dispatch and the optimal power flow problems discussed in the introduction. In addition to them, the problem of control in one dimension is also discussed. Chapters 5 and 6 represent the theoretical content of the work, in which the algorithm is presented and discussed, and the methodology of explainability is enunciated. Chapter 7 is the largest in extension, with the discussion and the results of the experiments. Chapter 8 closes the work with a conclusion of the text and a conversation on future work.

# Chapter 2

# Related Work

In this chapter we have a summary of work related to what is proposed in this writing. As the proposed algorithm is found in the intersection of traditional models and the use of neural networks, this survey is divided into two parts: operational research models for the MSSOs and neural models that try to solve optimization problems.

## 2.1 State-of-the-Art

There is no algorithm that guarantees convergence for any type of MSSO. Here we will discuss three aspects of the algorithms in literature: the algorithms that solve convex subproblems, the algorithms that solve subproblems with convex restrictions and objective function but with integer variables and the problems with time dependence. Our algorithm, SNDP, seeks to be able to solve arbitrary subproblems (as long as there is a solver for such a type of problem), possibly with time dependence.

### 2.1.1 Convex Subproblems

MSSO algorithms usually suffer from an exponential behavior in which the number of decision variables grows exponentially with the number of stages [23]. This factor limits the use of these algorithms to only low-dimensional instances. As previously said, the SDDP solves this problem by proposing a breakdown by stage and learning a future cost function for each, without having to list all decision variables anymore [86].

Unfortunately, the worst case of SDDP still scales exponentially in the number of stages, thus choosing for simplifications to find solutions in an acceptable computing time [15, 49]. Among these simplifications, we have a sample stage of Monte Carlo to reduce

the number of variables of decisions to be considered [77], but that can lead to a over-adjustment in training data and a bad performance in test data. In [24], the authors introduced a neural model, Neural Stochastic Dual Dynamic Programming ($\upsilon$-SDDP), which allows the SDDP to act in a lower-dimensional space, improving the behavior in the test and the training time. Despite the similarity of the names, $\upsilon$-SDDP and SNDP have a little common. The first seeks to accelerate the performance of the traditional SDDP while the second seeks to solve a new class of problems with the use of neural networks.

Regarding the dispatch problem, it is common that the SDDP and other MSSO algorithms use synthetic rain scenarios to represent the flow of the rivers. These scenarios are usually of a periodic self-regressive model [69]. It is important to note that the problems become increasingly difficult to solve as the number of these scenarios grows. Another challenge is that the effectiveness of these algorithms depends largely on how well these synthetic scenarios represent real inflows. To generate small samples of scenarios representing the stochastic process, variants of the Optimal Scenario Reduction method were proposed [81] to cut branches of the tree of the self-regressive models. However, these hydrological models are being challenged by the impacts of climate change [29]. This is an important limitation that can damage the performance of these algorithms in test data, since the scenarios can differ from the real.

In [32], the authors proposed robust, or risk-averse, formulations to protect the system from critical hydrological scenarios. The formulation uses a combination of SDDP and Benders decomposition to ensure that the policy obtained guarantees levels of the reservoirs that are high enough to protect the system from these critical scenarios.

### 2.1.2 Integer Subproblems

By going from linear subproblems to integer ones we have two consequences: future cost functions may not be convex anymore [115] and reduced costs are no longer subgradients [76]. As it is not easy to adapt the decomposition strategies to non-convex functions, to solve these problems, that is, MSSOs with integer variables, new approaches are required. Three of them stand out: Stochastic Dual Dynamic Integer Programming (SDDiP), Mixed Integer Dynamic Approximation Scheme (MIDAS) and Stochastic Lipschitz Dynamic Programming (SLDP).

SDDiP [115] addresses non-convexity by reformulating the subproblems in each stage with a new class of cuts, the Lagrangian cuts. The authors demonstrate that for problems in which states are binary variables, the algorithm has convergence in a finite

number of iterations with a probability equal to 1.

The algorithm follows in a similar way to the SDDP, with a forward pass and the backward pass. In the forward pass, the policy learned so far is simulated to have a sample of the most visited regions in the state space. In the backward pass, the algorithm updates the estimates of future cost on the states shown in the previous step by adding the Lagrangian cuts.

In optimization problems, lagrangian relaxation [41] is a method that approaches a difficult problem by one simpler, but the solution of second is still valid for first. The idea is to relax some of the restrictions by penalties, the lagrangian multipliers. The problem, thus, can be easier to solve, but we no longer have the guarantees of optimality.

The Mixed Integer Dynamic Approximation Scheme (MIDAS) [89] also has an approach similar to the SDDP. The idea is to approximate the future cost function as a step function instead of using cuts. The state space is divided into a finite number of regions and the local value of the function is approximated as a constant. There is still the possible extension of using cuts in these regions rather than a constant.

This approach has the premise that the future cost function is monotonous, or approximately monotonous, in the state variables. Each subproblem is formulated as a mixed integer programming and solved up to optimality. These subproblems are discretizations of the actual future cost function, and become more accurate as the resolution increases.

Stochastic Lipschitz Dynamic Programming (SLDP) [3] is another strategy that follows the SDDP archetype. Instead of cutting plans, lagrangian cuts or step functions, the authors propose the use of Lipschitz cuts, a kind of non-linear cuts. These cuts iteratively sub-approximate the future cost function until convergence is obtained. The studied Lipschitz cuts families are modeled as mixed integer programming so the class of the original problem is not changed. There are two Lipschitz cuts families studied:

- Reverse-Norm Function: cuts based on the idea of penalizing the norm of the gradient of the future cost function;
- Augmented Lagrangian Cuts: cuts obtained by the augmented norm of the lagrangian duality, which involves adding a square penalty to the lagrangian function. The dual problem obtained can be solved by optimization techniques with subgradient.

### 2.1.3 Time Dependency

In the SDDP algorithm there is the assumption that the uncertainty of different stages are independent of each other. This limits the types of problems that can be solved. Two approaches are used in literature, autoregressive processes and markovian modeling. In an expedition experiment in the electric system of Brazil, the policy learned by the first was dominated by the second [67].

If uncertainty comes from a autoregressive process, it is possible to incorporate it into the original algorithm without modifications [87], provided that the observations appear additively in the objective function and in the constraints. For this, it is enough to consider the past observation as a state variable, and the uncertainty of the stage is only the noise of the process.

Another approach, but that has no guarantee of convergence, is to model uncertainty as a markovian process [88]. The probability of observing a value depends on the state that the system is located, and that state evolves following a Markov chain. The strategy is, for each stage, to learn a different collection of cuts for each state of the chain.

## 2.2 Neural Networks for Optimization Problems

There is an interest in incorporating recent advances in deep learning to solve problems of optimization. Two main approaches have emerged so far: the use of neural networks to estimate the optimal policies of problems of control; and the use of neural networks with external solvers to obtain semantic representations of raw data. The approach of this work, which will be discussed in subsequent chapters, is in the intersection of both, by using neural networks to estimate policies for problems of control, but with the help of external solvers.

### 2.2.1 Deep Reinforcement Learning

Reinforcement Learning is the problem of an agent having to learn a behavior through interactions, or observations of interactions, with a dynamic environment [55].

There are two main classes of methods to deal with reinforcement learning:

- **Actor**: searching through the policy space to find the one that works best in the environment (e.g., genetic algorithms and genetic programming);
- **Critic**: use statistical and dynamic programming techniques to estimate the usefulness of taking certain actions, and create a policy that maximizes that usefulness.

There are also algorithms in the intersection of these two classes, the Ator-Critic models [60], in which the critic is a neural network used to estimate the utility function, and this is used to modify the parameters of a second neural network, the actor.

It is common to reinforcement learning algorithms to work on Markov's Decision Processes (MDP) [11]. Similar to MSSO, MDPs provide a useful conceptual structure to capture the behavior of an optimal policy. A MDP is defined by 4 objects:

- A set $S$ of states, called the state space;
- A set $A$ of actions, called the action space;
- A transition probability function $P_a(s, s') = \mathcal{P}(s_{t+1} = s' | s_t = s, a_t = a)$ of an action to transition the stae $s$ to $s'$ on the next stage;
- A immediate reward function $r = R_a(s, s')$ of the transition from state $s$ to $s'$ by action $a$.

In this context, a policy is a mapping from any state $s \in S$ to actions $a \in A$. In MDPs, the goal is then to find a policy that maximizes a cumulative function of the rewards, typically the sum adjusted by a discount factor $\gamma$:

$$\rho = r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots = \sum_{t=0}^{\infty} \gamma^t r_t. \tag{2.1}$$

In reinforcement learning, the probabilities are unknown and not modeled explicitly, but rather accessed through simulations of an environment.

Policy Gradient (PG) [104] is a technique to seek for a policy in reinforcement learning. Instead of trying to approximate a utility function and use it to compute the policy, PG seeks for a stochastic policy $\pi_\theta$ directly using a function approximator with its own parameters. For example, a neural network with the representations of states as inputs and the probability of actions as outputs. Be $\theta$ the vector of parameters and $\rho_\gamma$ the geometric sum of rewards, then the parameters are changed by:

$$\Delta\theta \approx \alpha \frac{\partial \rho_\gamma}{\partial \theta} \tag{2.2}$$

where $\alpha > 0$ is the step length. If an estimate of this gradient can be obtained, it is likely that we have $\theta$ converging to a local optimum of the policy space. For this, PG methods provide the ascending gradient algorithm $\hat{g}$ estimates of the $\theta$ gradients, commonly in the form:

$$\hat{g} = \mathbb{E}_t \left[ \nabla_\theta \log \pi_\theta(a_t|s_t) \hat{A}_t \right], \tag{2.3}$$

in which $\hat{A}_t$ is an estimator of the advantage function (i.e., how good is an action compared to the average) at time $t$. Here the expectation $\mathbb{E}_t[\ldots]$ is an empirical average on a finite number of samples. This estimate is obtained by differentiating an objective function of the way:

$$L(\theta) = \mathbb{E}_t \left[ \log \pi_\theta(a_t|s_t) \hat{A}_t \right]. \tag{2.4}$$

Proximal Policy Optimization (PPO) [98] is a family of policy gradients methods for reinforcement learning proposed by OpenAI, which alternates between sampling data when interacting with the environment, and optimizing a target function using gradient ascent. By its performance, PPO has become the state-of-the-art [109]. The difference is given by the target function chosen.

Let $\theta_{\text{old}}$ be the parameter vector before an iteration, and let $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$. Since $r_t(\theta_{\text{old}}) = 1$, we can measure the degree of change between updates. The objective function is

$$L(\theta) = \mathbb{E}_t \left[ \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}(a_t|s_t)}} \hat{A}_t \right] = \mathbb{E}_t \left[ r_t(\theta) \hat{A}_t \right], \tag{2.5}$$

which, if unconstrained, can lead to a too large update of the to-be-maximized policy. The question is how to modify said function in order to avoid large diversion of $r_t(\theta)$ in relation to 1. The authors propose

$$L^{CLIP}(\theta) = \mathbb{E}_t \left[ \min\{ r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \} \right], \tag{2.6}$$

Where $\epsilon$ is a hyperparameter. We have that the first term is still the previous objective function, and the second modifies the function by cutting the probability ratio, which removes the incentive to move $r_t(\theta)$ out of the $[1-\epsilon, 1+\epsilon]$ interval. As we have the minimum of the two terms, the result is a lower limit of the non-cutting goal.

With this objective function, we can describe PPO as the algorithm 1, in a Actor-Critic style:

This algorithm reminds the one that will be proposed, in which we have a step of policy simulation and a step of learning. But under the established reinforcement learning perspective, our algorithm would only be a critic.

**Algorithm 1:** Proximal Policy Optimization

```
for iteration= 1, 2, ... do
    for actor= 1, 2, ..., N do
        Execute the policy π_old on the environment for T steps;
        Compute the advantage estimates Â_1, ..., Â_T;
    end
    Optimize the objective function L^CLIP(θ) w.r.t. θ, with K epochs and
     mini-batches of size < NT;
    θ_old ← θ;
end
```

### 2.2.2 Gradients of Optimizations

Instead of using the neural networks as solvers, as in the case of deep reinforcement learning, there is a second approach in which the neural network is connected to an external problem solver. The idea is that the solver acts as a layer of the network, in which previous layers act on the input data and provide for the first a purely semantic data. The difficulty is in how to approximate a gradient to the output of these solver.

Disciplined convex problems [45] are a subclass of convex optimization problems, in which a set of conventions are imposed on the model in order for their solution process to be automated. These problems are efficiently differentiable, which allows you to use gradient descent to optimize policies on control problems [2].

For problems with linear objective functions, such as the problem of the minimum path [30], the traveling salesman problem [36] and the perfect matching of minimum cost problem [28], a gradient approximation was proposed by disturbing the optimization entries and interpolating the results [90]. Thus, the authors were able to extract useful intermediate representations for the optimizations, for example to find the minimum path in digital gaming terrains by estimating the locomotion costs.

For optimization problems in combinatorics, we can define a distribution of a discreet exponential family over the solution space, in which the optimal solution is the estimator of maximum likelihood. This distribution is differentiable [79] in a way similar to the proposed in [90], however the results have superior convergence in experiments.

There are also specific approaches, in which neural networks are used in particular algorithms, such as A* [112]. One extension [5] is to combine the general [90] and specific [112] approaches to optimize at the same time the edges weights and the heuristic. Experimental results showed improved performance over both separately.

The idea of these algorithms is therefore not to solve an arbitrary optimization problem and more to transform the input data into a context in which optimization

makes sense. This differs from the challenge to be dealt with in this work, but shows that there is ways to combine the neural networks with the solvers. If these work seek to find the gradients of the solvers, we will use the solvers as a target in supervised learning.

# Chapter 3

# Theoretical Foundations

In this chapter we have the definition of the SDDP algorithm, the basis of our modeling, as well as the other theoretical contextualization necessary for the understanding of the proposed methodologies.

## 3.1 Manifold Geometry

In this section we describe the main properties of manifolds necessary for our work. A manifold is a space that locally resembles an euclidian space. A $d$-dimensional manifold resembles locally $\mathbb{R}^d$. By resembling it is understood that each point has a homeomorphic neighborhood, that is, there is a map between the space and an open set in the vector space that preserves the topological properties. The concept of manifold was first studied in [92]

In this text we will only work with manifolds that meet $\mathcal{S} = \{(x, f(x)), \quad \forall x \in \mathfrak{D}\}$ in which $f : \mathbb{D} \to \mathbb{R}$ is a continuous function and $\mathfrak{D}$ is its domain, and therefore the use of the term will often refer only to that subset of varieties. Spheres for example are manifolds but, for the purpose of this work, won't be considered.

### 3.1.1 $K$-Convex Manifolds

Let $f : \mathbb{R}^n \to \mathbb{R}$ be any function. It is called **convex** iif it satisfies

$$\forall \quad 0 \leq t \leq 1 \text{ e } x_1, x_2 \in \mathbb{R}^n : f(tx_1 + (1-t)x_2) \leq tf(x_1) + (1-t)f(x_2). \tag{3.1}$$

Intuitively the above condition says that any segment that unites two points in the

image of $f$ is in the image itself, or is "above" it. A consequence of the definition is that any local minimum of a convex function is also a global minimum.

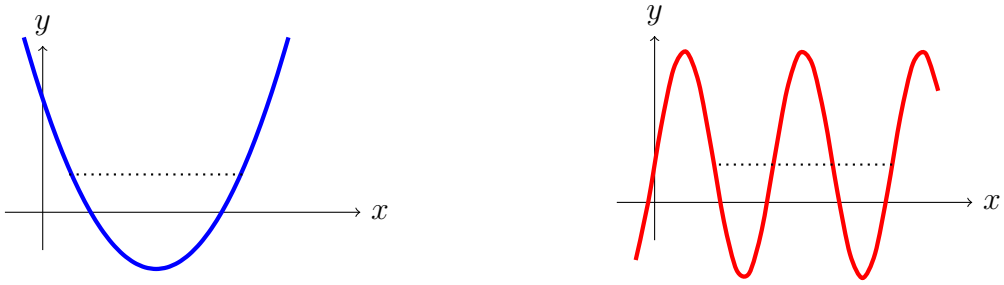


Figure 3.1: Graphic example of a convex and a non-convex function.

An example of an important convex function for our work is a set of cuts, in which every cut is an affine function on its arguments and the image of the function is the maximum of all cuts, $f(\mathbf{x}) = \max\{\beta_i \mathbf{x} + \alpha_i : i = 1, ..., n\}$. An example is given in figure 3.1.1.

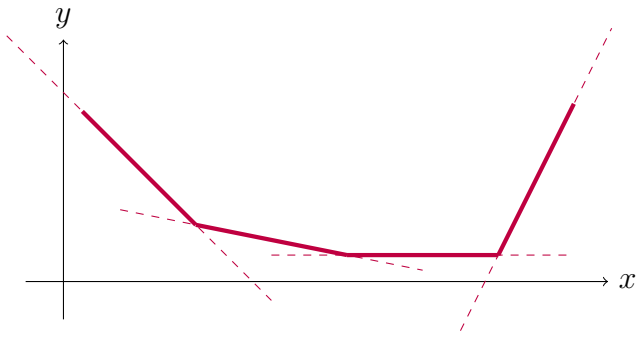


Figure 3.2: Graphical example of a set of cuts function.

The functions which don't satisfy this definition are called **non-convex**. We will also consider the subset of the non-convex function that are the grouping of a finite number of convex functions. A function $f$ is called **k-convex** if

- $\exists \quad \mathcal{F}_k = \{g_1, ..., g_k\}, g_i$ is convex, such that $f(x) = \min\{g_i(x) : g_i \in \mathcal{F}_k\}$;
- $\nexists \quad \mathcal{F}_{k-1} = \{g_1, ..., g_{k-1}\}, g_i$ is convex, such that $f(x) = \min\{g_i(x) : g_i \in \mathcal{F}_{k-1}\}$.

We extend the convexity attribute (convex, non-convex, $k$-convex) of a function to the associated manifold.

### 3.1.2 Piecewise Linear Function

A piecewise linear function is a real function in which its graphical representation is defined by affine segments (lines, planes, hyperplanes etc.), like figure 3.3. For a formal definition, we'll use the approach by [83].

Be $\Gamma$ a convex closed domain in $\mathbb{R}^n$. A function $f : \Gamma \to \mathbb{R}$ is piecewise linear if there is a finite family $\mathcal{Q}$ of closed domains such that $\Gamma = \cup\mathcal{Q}$ and $f$ is affine in all domains within $\mathcal{Q}$. A unique affine function $g(\mathbf{x}) = \mathbf{a} \cdot \mathbf{x} + b$ in $\mathbb{R}^n$ that coincides with $f$ in a given $Q \in \mathcal{Q}$ is called a component of $f$.

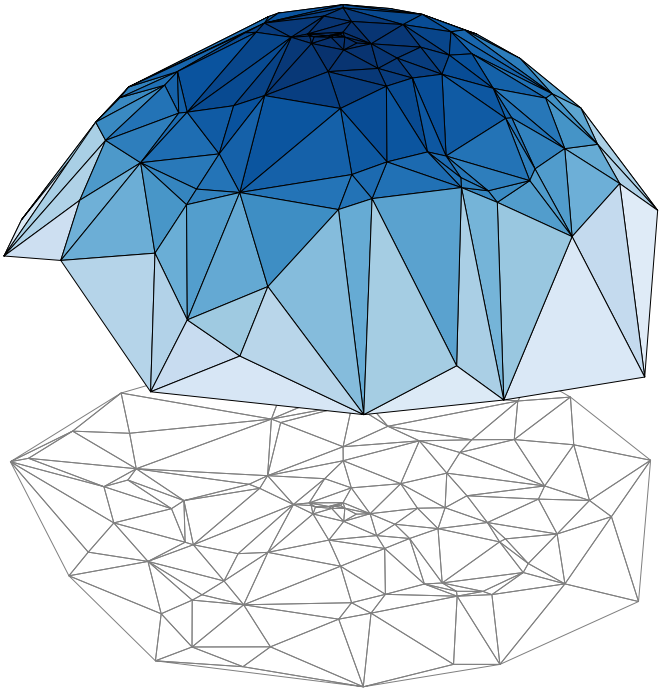

Figure 3.3: 2D piecewise linear function (top) and its domains $Q \in \mathcal{Q}$ in which it is affine (bottom).

In [83] we have the proof for the following theorem. ($a$) Let $f$ be a piecewise linear function in $\Gamma$ and $\{g_1, ..., g_n\}$ its set of unique components. There exists a family $\{S_j\}_{j \in J}$ of non-empty subsets $\{1, ..., n\}$ such that

$$f(\mathbf{x}) = \min_{j \in J} \max_{i \in j} g_i(\mathbf{x}). \tag{3.2}$$

$b$ Reciprocally, for any family of distinct affine functions $\{g_1, ..., g_n\}$, the above equation define a piecewise linear function.

In this work, we will not explicitly model piecisewise linear functions in a closed domain. However, for all the problems discussed it will be possible to limit the set of solutions to a closed convex domain large enough.

In particular for this work, we will study the convex and $k$-convex piecewise linear functions. If we consider that $J$ has only one set, set $j$, the equation 3.2 becomes the maximum of affines functions. By the definition of convex function given by the equation 3.1 it is possible to observe two properties:

(a) Affine functions are convex;

(a) The maximum of two convex functions is also a convex function.

Thus, a set $j$, which we will call a set of cuts, defines a convex piecewise linear function. It is easy to optimize such functions by modelling it as a linear program

$$\begin{aligned} \min_{\mathbf{x}\in\mathbb{R}^n} \quad & \theta & & \\ & \theta \geq \mathbf{a_i} \cdot \mathbf{x} + b_i & & i \in j \end{aligned} \tag{3.3}$$

By the uniqueness of the functions $g$ and as $\emptyset \notin J$, we know that if $|J| = k > 1$ then the function $f$ corresponds to a $k$-convex piecewise linear function, with $k$ distinct sets of cuts. As $f$ is a non-convex function, it is not possible to minimize it as a linear program, being necessary the use of integer variables.

$$\begin{aligned} \min_{\mathbf{x}\in\mathbb{R}^n} \quad & \omega & & \\ & \theta_j \geq \mathbf{a_i} \cdot \mathbf{x} + b_i & & i \in S_j, 1 \leq j \leq k \\ & \omega \leq \theta_j & & 1 \leq j \leq k \\ & \omega \geq \theta_j - M \cdot (1 - y_j) & & 1 \leq j \leq k \\ & \sum_{j=1}^{k} y_j = 1 & & \\ & \mathbf{y} \in \{0,1\}^k & & \end{aligned} \tag{3.4}$$

where $M$ is a sufficiently large number. Note that $k$ integer variables are necessary for such approach. The difficulty to solve it grows with the number of sets of cuts.

Neural networks with ReLU activation function are also non-conve piecewise linear functions, as will be seen in the section 3.1.4. In its optics however, we don't have the list of all the affine regions it model, being not possible to use the same modeling 3.4. A new approach is required.

### 3.1.3 $k$-Convex Piecewise Linear Function Fitting

The optimization of a $k$-convex piecewise linear function fitting through gradient descent will be given in the same way of [27]. In this work, the authors are interested in obtaining the decomposition of 3 dimensional structures in $k$ convex hulls. Note that the difference between convex hulls and convex functions is the use or not of the signal function.

Be $\mathcal{H}_h(\mathbf{x}) = \mathbf{a}_h \cdot \mathbf{x} + b_h$ the cut $h$ of a set of cuts. Given a sufficiently large number $H$ of cuts, any convex function can be approximated by taking the maximum of the cuts. To facilitate learning, instead of the maximum, we will use a soft maximum function, LogSumExp, and the estimated function is defined as:

$$\Phi(\mathbf{x}) = \log \sum_{h=1}^{H} \exp\left(\sigma \mathcal{H}_h(\mathbf{x})\right), \tag{3.5}$$

In which the parameter $\sigma$ controls the smoothness of the convex function learnt. Thus, it is easy to add this convex approximation as an output layer of a neural network. As seen in equation 3.2, with $k$ convex approximations like this we can have a $k$-convex approximation if we take the minimum of all of them.

### 3.1.4 ReLU Neural Network as a Piecewise Linear Function

In this section, we'll discuss how neural networks with ReLU activation function are piecewise linear functions. The number of linear regions grows exponentially in the number of layers of the network, but their minimization requires only a number of integer variables equal to the number of neurons of the network. However, empirically, in the chapter 7, we see that the depth of the network affects the search time of the minimization.

If it is a ReLU network, i.e., a sequence of compositions of affine functions and ReLU activations, to show that the network is also a piecewise linear function, it is enough to show that:

(a) A affine combination of piecewise linear functions is also a piecewise linear function;

(b) The composition of ReLU with a piecewise linear function is also a piecewise linear function.

With the definition a piec3ewise linear function in equation 3.2, it is easy to prove the first property:

$$\begin{aligned}\sum_{k=1}^{n} \lambda_k f_k(\mathbf{x}) &= \sum_{k=1}^{n} \lambda_k \min_{j\in J_k} \max_{i\in j} g_i(\mathbf{x}) \\ &= \sum_{k=1}^{n} \min_{j\in J_k} \max_{i\in j} \lambda_k g_i(\mathbf{x}) \\ &= \min_{j_1\in J_1,\dots,j_k\in J_k} \sum_{k=1}^{n} \max_{i\in j_k} \lambda_k g_i(\mathbf{x}) \\ &= \min_{j_1\in J_1,\dots,j_k\in J_k} \max_{i_1\in j_1,\dots,i_k\in j_k} \sum_{k=1}^{n} \lambda_k g_{i_k}(\mathbf{x}) \\ &= \min_{j_1\in J'} \max_{i\in j} g'_i(\mathbf{x})\end{aligned} \tag{3.6}$$

The second property is also self-evident. Let us define $g_0(\mathbf{x}) = 0$. Then:

$$\begin{aligned}\mathrm{ReLU}(f(x)) &= \max\{0, \min_{j\in J_k} \max_{i\in j} g_i(\mathbf{x})\} \\ &= \min_{j\in J_k} \max_{i\in j\cup\{0\}} g_i(\mathbf{x})\end{aligned} \tag{3.7}$$

Thus, neural networks with ReLU activation functions are piecewise linear functions.

As we know, the response of a neuron (affine function on the inputs and ReLU activation) is a function $\mathbb{R}^n \to \mathbb{R}$ piecewise linear. Thus, when we compose a new piecewise linear function in that output, i.e. a new layer of the network, we create new linear regions that take advantage of the geometry of the previous ones. An analogy used by [74] is that of thinking in the linear regions as foldings, such as those of paper, in the space of the input data. Thus the composition of layers in the network would represent a sequence of foldings, as illustrated in 3.4.

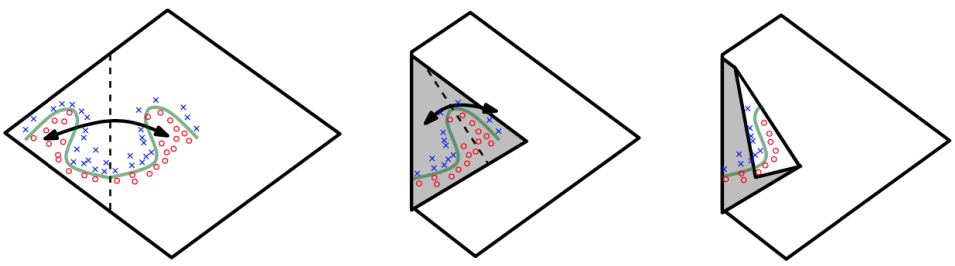

Figure 3.4: Non-trivial space folding, in 2D. The folding can potentially identify hidden symmetries. Figure by [74].

A major corolary of [74] is described below. A neural network with ReLU activation function with $n_0$ input neurons and $L$ layers with width $n \geq n_0$ can compute

functions with $\Omega\Big((n/n_0)^{(L-1)n_0}n^{n_0}\Big)$ linear regions. Thus, the number of linear regions grows exponentially in the number of layers $L$ and in a polynomial way in the number of neurons per layer $n$. In addition, the corolary can be described according to the number of parameters, revealing a similar result, the parameters are exponentially more efficient than a shallow network. Note that a shallow neural network with $Ln$ neurons is the equivalent to a $\Omega(Ln)$-convex manifold. So, as we will see in the modeling below, the number of binary variables, a heuristic for the difficulty of solving an integer program, is exponentially more efficient since it scales with the number of neurons.

In much of this work, neural networks have been used as functions estimators of the type $f : \mathbb{R}^n \to \mathbb{R}$, non-analytical, associated with the cost of a decision. In order to find the lowest cost decision, it is necessary to minimize these estimators. Multi-layer perceptron neural networks with activations of the type Reduced Linear Unit (ReLU) [38] can be represented as integer programming problems and thus easily incorporated into the solvers of this class of problem. Be $\mathbf{W}^{(l)}$ and $\mathbf{b}^{(l)}$ the weights and biases of the layer $l$, the problem of minimizing a given network of said type is:

$$
\begin{aligned}
\min_{\mathbf{x}\in\mathbb{R}^n} \quad & \mathbf{h}^{(L)} && && (3.8)\\
& \mathbf{a}^{(0)} = \mathbf{x}, && \\
& \mathbf{a}^{(l)} = \mathbf{W}^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)} && 1 \leq l \leq L,\\
& \mathbf{h}^{(l)} \geq \mathbf{a}^{(l)} && 1 \leq l \leq L,\\
& \mathbf{h}^{(l)} \leq M^{(l)} \cdot \mathbf{y}^{(l)} && 1 \leq l < L,\\
& \mathbf{h}^{(l)} \leq \mathbf{a}^{(l)} + M^{(l)} \cdot (1 - \mathbf{y}^{(l)}) && 1 \leq l < L,\\
& \mathbf{h}^{(l)} \geq 0 && 1 \leq l \leq L,\\
& \mathbf{a}^{(l)}, \mathbf{h}^{(l)} \in \mathbb{R}^{n^{(l)}} && 1 \leq l \leq L,\\
& \mathbf{y}^{(l)}, \in \{0,1\}^{n^{(l)}} && 1 \leq l \leq L,
\end{aligned}
$$

in which $M^{(l)}$ is a sufficiently large number for layer $l$.

## 3.2 Stochastic Dual Dynamic Programming

Stochastic Dual Dynamic Programming (SDDP) is an algorithm for solving multi-stage Stochastic optimization problems. By multi-stage it is understood as problems in which an agent takes a sequence of decisions over time. By Stochastic, it is understood

that the agent is making decisions under the presence of uncertainty that is gradually revealed over the stages.

### 3.2.1 Algorithm's Assumptions

For the modeling of systems under the SDDP structure, three types of variables are used:

- **State Variables**: variables that track a system property over time (e.g., reservoir volumes, number of items in an inventory, spatial position of a vehicle etc.). We'll use the $x$ notation to describe them.
- **Control Variables**: the actions taken at each stage that modify the state variables (e.g., water outflow from the reservoirs, buying stock, vehicle acceleration etc.). We'll use the $u$ notation to describe them.
- **Random Variables**: external random variables observed at the beginning of a stage, before the control. (e.g., rain inflow, item defects, steering errors etc.). We'll use the $\omega$ notation to describe them and its sample space at stage $i$ as $\Omega_i$.

At each stage, all three variables, $x, u$ e $\omega$, relate to each other by a transition function which maps the previous state, the controls and the random variables to a new state such that $x' = T_i(x, u, \omega)$. This transition has a cost $C_i(x, u, \omega)$, called stage objective. The agent chooses its actions following a policy $u = \pi_i(x, \omega)$, which must satisfy some constraints $u \in U_i(x, u, \omega)$.

The problem space that can be modeled under this structure is too large for an algorithm to find solutions in a treatable way for all. SDDP, thus, has some assumptions:

1. Finite number of stages;
2. Finite and discrete random variables;
3. For a fixed $\omega$, $C_i$ is a convex function, $T_i$ is linear and $U_i$ is a closed set.

### 3.2.2 Recursion and Subproblems

The next step is to compute the policy. A method is to use the Bellman principle of optimality [10], the dynamic programming, and define the recurrent subproblem as:

$$\begin{aligned} V_i(x,\omega) = \min_{\bar{x},x',u} \quad & C_i(\bar{x},x',u) + \mathbb{E}_{j\in i^+,\phi\in\Omega_j}\left[V_j(x',\phi)\right] \\ & x' = T_i(\bar{x},u,\omega) \\ & u \in U_i(\bar{x},\omega) \\ & \bar{x} = x. \end{aligned} \tag{3.9}$$

These problems are difficult to solve accurately, as they involve recurrent optimization problems with many nested expectations. Instead of solving them accurately, SDDP iteratively approaches the expectations of each subproblem with a future cost function. Thus, we can think of the problem as a sequence of subproblems and for now abstract these expectations.

### 3.2.3 Kelley's Cutting Plane Algorithm

Kelley's cutting plane algorithm is an iterative method for convex function minimization. Given a convex function $f(x)$, the algorithm builds lower approximations to the function at its minimum $f$ through a set of first-order Taylor expansions over the points $x_1, ..., x_K$, called cuts:

$$\begin{aligned} f^{(K)} = \min_{\theta\in\mathbb{R},x\in\mathbb{R}^N} \quad & \theta \\ & \theta \geq f(x_k) + \frac{d}{dx}f(x_k)^\top (x - x_k), \quad k = 1,\dots,K \\ & \theta \geq M, \end{aligned} \tag{3.10}$$

in which $M$ is an inferior bound for the value of $f(x)$ for all $x$ in the domain.

By convexity, $f^{(K)} \leq f(x)$ for all $x$. If $x^*$ is the point which minimizes the function, then we have that $f^{(K)} \leq f(x^*) \leq \min_{k=1,\dots,K} f(x_k)$. If that difference is small enough, we can conclude our search e declare that we've found a solution close to the optimum.

the algorithm is structured in such a way that $\lim_{K\to\infty} f^{(K)} = \min_{x\in\mathbb{R}^N} f(x)$.

**Algorithm 2:** Kelley's Cutting Plane

**Data:** $f(x)$ convex, $K_{max} > 0, \epsilon > 0$
**Result:** Approximation for the minimum $f$
$ls \leftarrow \infty$;
$li \leftarrow -\infty$;
$K \leftarrow 0$;
Initialize $f^{(K)}$;
**while** $K \leq K_{max}$ $e$ $|ls - li| \geq \epsilon$ **do**
  Solve $f^{(K)}$ and obtain a candidate solution $x_{K+1}$;
  $ls \leftarrow \min\{ls, f(x_{K+1})\}$;
  $li \leftarrow f^{(K)}$;
  Add cut $\theta \geq f(x_{K+1}) + \frac{d}{dx} f(x_{K+1})^\top (x - x_{K+1})$ to create $f^{(K+1)}$;
  $K \leftarrow K + 1$;
**end**

### 3.2.4 Future Cost Function Approximation

Note that in the subproblem formulation, if we exclude the expectations of future stages costs, we have a convex program. As $x$ appears only in the constraint $\bar{x} = x$, $V_i(x, \cdot)$ is convex in $x$ to a fixed $\omega$. In addition, this constraint is important because its reduced cost, or opportunity cost, is the subgradient of $V_i$ with respect to $x$. Thus, we can use Kelley's cutting plane algorithm to minimize the functions of $V_i$ in a treatable way:

$$
\begin{aligned}
V_i(x, \omega) = \min_{\bar{x}, x', u} \quad & C_i(\bar{x}, x', u) + \theta \\
& x' = T_i(\bar{x}, u, \omega) \\
& u \in U_i(\bar{x}, \omega) \\
& \bar{x} = x \\
& \theta \geq \mathbb{E}_{j \in i^+, \phi \in \Omega_j} \left[ V_j^k(x'_k, \phi) + \frac{d}{dx} V_j^k(x'_k, \phi)^\top (x' - x'_k) \right], \quad k = 1, \ldots, K \\
& \theta \geq M.
\end{aligned}
\tag{3.11}
$$

What is missing is a method to generate these cuts iteratively, and for this we need candidate solutions $x'_k$. One possibility is to generate viable candidates uniformly at random. The problem is that these samples may not represent situations of the state variables that occur in practice. A better way to generate these candidates is by simulating the policy. The SDDP does this in two stages:

- **Forward pass**: we simulate the policy stage by stage, from the beginning to the end.

At each stage, a realization of the random variable is observed and the approximate subproblem is solved to generate a candidate for the next stage.

- **Backward pass**: To add the new cuts, visit the stages in inverse order, so that by refining a stage, we will already have refined its successor.

Thus, SDDP works in a similar way to what happens in the deep learning area's backpropagation algorithm: the forward pass pushes primal information along the stages (state variables), and the backward pass brings the dual information (cuts) back to the stages to improve the decisions for the next iteration.

## 3.3 Surface Curvature

This section presents the basic concepts on the surface (two dimensional manifolds) curvature theory, as well as the methods used to estimate computationally. Especial attention will be given to the attributes derived from the principal curvatures.

### 3.3.1 Curvature Primer

Much of the study on surface curvature was given by Gauss' work in studying the total curvature of a point (which today is called Gaussian Curvature) according to the maximum and minimum curvature [39]. These concepts are important for the proposed study of the future cost functions of the dynamic programming algorithms and therefore will be elucidated below. The notation and intuition of these will be given differently from the original version of Gauss, in order to facilitate understanding.

Let's begin with the curvature of differentiable curves in a euclidean space. The osculating circle of one point in the curve, the kiss circle in Latin [63], is the limit of the circles defined by three points in the curve that approach the original point. The figure 3.5 shows how the oscillating circle is what separates the circles on one side and on the other of the curve, at a given point. We will define the curvature of one point in the curve as the inverse of the radius of the osculating circle.

The curvature can also be expressed in terms of the first and second derivatives [46]. Let $\gamma$ be a parametric curve in a $\mathbb{R}^n$ space, its curvature is given by:

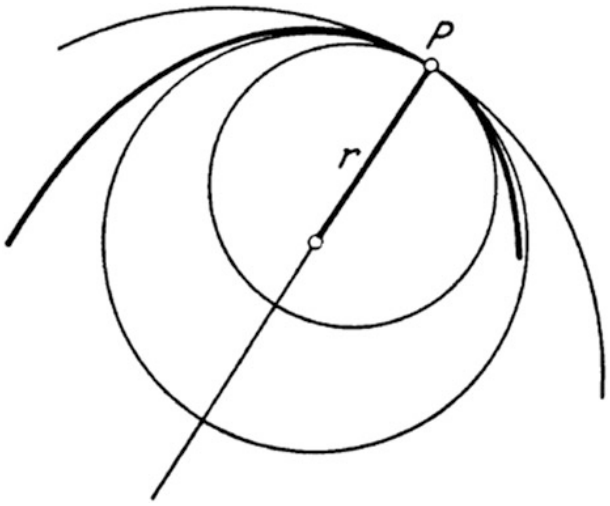


Figure 3.5: The osculating circle in P (of radius $r$) separates two types of circles tangent to the curve P (bold): those who are on one side of the curve from those who are on the other. Figure by [52].

$$\kappa = \frac{\sqrt{\|\gamma'\|^2\|\gamma''\|^2 - (\gamma' \cdot \gamma'')^2}}{\|\gamma'\|^3}. \tag{3.12}$$

Let us now consider differentiable surfaces. For each point on the surface we have a normal vector. A normal hyperplane will be the one that contains the normal vector. Its intersection with the surface will define a curve, in which we can calculate the curvature as we saw above. Note that as we have infinite normal hyperplanes, each in one direction, we can possibly have infinite different values of curvature to a point. This set of curvatures is called normal curvatures. The principal curvatures, $\kappa_1$ and $\kappa_2$, are those of greater and smaller absolute curvature value among the normal curvatures. The concept of **total curvature** proposed by Gauss, which will be called gaussian curvature, is defined as the product $\kappa_1 \cdot \kappa_2$. The mean of both principal curvatures will be called **mean curvature**.

In surfaces, the principal curvatures define local behavior, described in table 3.1 below:

| | | $\kappa_1$ | | |
|---|---|---|---|---|
| | | $< 0$ | $= 0$ | $> 0$ |
| $\kappa_2$ | $< 0$ | Concave Ellipsoid | Concave Cylinder | Hyperboloid Surface |
| | $= 0$ | concave Cylinder | Plane | Convex Cylinder |
| | $> 0$ | Hyperboloid Surface | Convex Cylinder | Convex Ellipsoid |

Table 3.1: Local behavior of surfaces by its principal curvatures.

Thus, to know the principal curvatures allows us to understand the local behavior around a point.

### 3.3.2 Numerical Computation of Curvature

There are mny ways to compute the curvatures of surfaces sampled in grids [103, 37]. We will use the method described in [93], where is performed a quadratic approximation of a point by its neighbors in the grid. Considering only a $3 \times 3$ neighborhood, like in 3.6, this approximation is reduced to simple analytical equations:

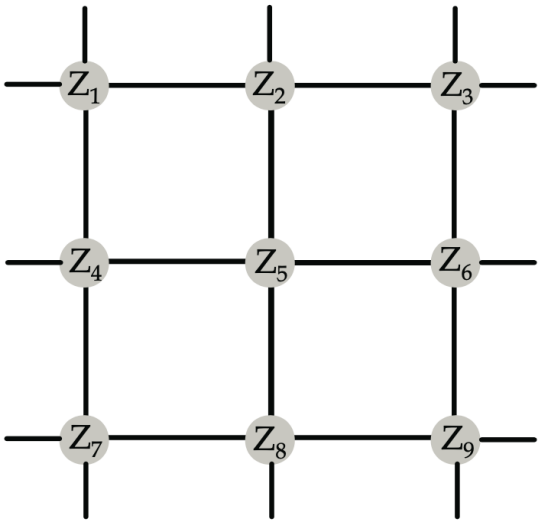


Figure 3.6: Example of a $3 \times 3$ grid centered at $Z5$. This is the neighborhood used to compute the curvatures. Figure by [93].

$$z = ax^2 + by^2 + cxy + dx + ey + f \tag{3.13}$$

$$a = \frac{1}{2}\frac{\partial^2 z}{\partial x^2} = \frac{Z_1 + Z_3 + Z_4 + Z_6 + Z_7 + Z_9}{12\Delta x^2} - \frac{Z_2 + Z_5 + Z_8}{6\Delta x^2} \tag{3.14}$$

$$b = \frac{1}{2}\frac{\partial^2 z}{\partial y^2} = \frac{Z_1 + Z_2 + Z_3 + Z_7 + Z_8 + Z_9}{12\Delta x^2} - \frac{Z_4 + Z_5 + Z_6}{6\Delta x^2} \tag{3.15}$$

$$c = \frac{\partial^2 z}{\partial x \partial y} = \frac{Z_3 + Z_7 - Z_1 - Z_9}{4\Delta x^2} \tag{3.16}$$

$$d = \frac{\partial z}{\partial x} = \frac{Z_3 + Z_6 + Z_9 - Z_1 - Z_4 - Z_7}{6\Delta x} \tag{3.17}$$

$$e = \frac{\partial z}{\partial y} = \frac{Z_1 + Z_2 + Z_3 - Z_7 - Z_8 - Z_9}{6\Delta x} \tag{3.18}$$

$$f = \frac{2(Z_2 + Z_4 + Z_6 + Z_8) - (Z_1 + Z_3 + Z_7 + Z_9) + 5Z_5}{9}. \tag{3.19}$$

With these constants, it is easy to compute the gaussian and mean curvatures on the quadratic approximation:

$$\kappa_g = \frac{4ab - c^2}{(1 + d^2 + e^2)^2}, \tag{3.20}$$

$$\kappa_m = \frac{a(1 + e^2) - cde + b(1 + d^2)}{(1 + d^2 + e^2)^{3/2}}, \tag{3.21}$$

and then solving a system of equations to obtain the principal curvatures.

### 3.3.3 Curvature Attributes

As we have seen, the principal curvatures describe the local behavior of a surface. In the study of seismic data, geologists defined new curvature-based attributes that describe or identify other local properties [93]. This research process is done by specialists with a visual analysis. In this work, we will use the attributes for an automatic detection of anomaly points. Below we have a brief description and the formulation, based on the quadratic constants, of a selection of attributes:

- **First Derivative Attributes**: dip angle and azimuth angle are mainly used for border detection and drastic changes of values;

$$\text{Dip Angle} = \tan^{-1}\left(\sqrt{d^2 + e^2}\right) \tag{3.22}$$

$$\text{Azimuth Angle} = \tan^{-1}\left(\frac{e}{d}\right) \tag{3.23}$$

- **Most Positive and Most Negative Curvatures**: similar to the principal curvatures, but it considers the sign, which removes local shape information but accentuate drastic changes;

$$\kappa_+ = (a + b) + \sqrt{(a - b)^2 + c^2} \tag{3.24}$$

$$\kappa_- = (a + b) - \sqrt{(a - b)^2 + c^2} \tag{3.25}$$

- **Dip Curvature**: curvature at the direction of largest slope, exaggerate local relief [75];

$$\kappa_d = \frac{2(ad^2 + cde + be^2)}{(d^2 + e^2)(1 + d^2 + e^2)^{3/2}} \tag{3.26}$$

- **Strike Curvature**: curvature at the perpendicular direction of the largest slope, describes the morphology perpendicular to the slope [75];

$$\kappa_s = \frac{2(ae^2 - cde + bd^2)}{(d^2 + e^2)(1 + d^2 + e^2)^{1/2}} \tag{3.27}$$

- **Contour Curvature**: similar to strike curvature, but its magnitude is less well behaved, represents the curvature of the contour which passes through that point [75]

$$\kappa_c = \frac{2(ae^2 - cde + bd^2)}{(1 + d^2 + e^2)^{3/2}} \tag{3.28}$$

- **Shape Index**: combines the information of the principal curvatures in a single index to identify local behavior [59], like the table 3.1, see figure 3.7;

$$S_i = \frac{2}{\pi} \tan^{-1}\left(\frac{\kappa_2 + \kappa_1}{\kappa_1 - \kappa_2}\right) \tag{3.29}$$

- **Curvedness**: describes the magnitude of the curvature, with no attention for local shape [59].

$$\kappa_n = \sqrt{\frac{\kappa_1^2 + \kappa_2^2}{2}} \tag{3.30}$$

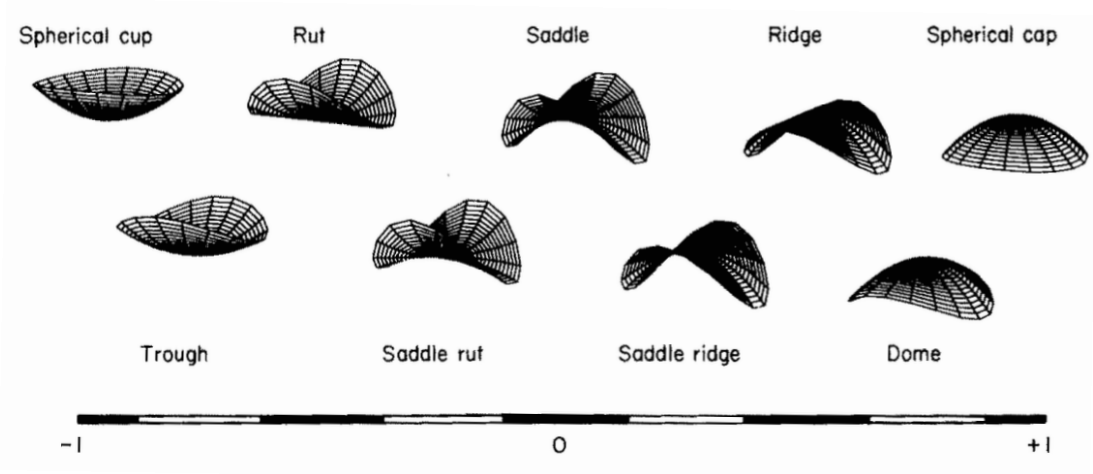


Figure 3.7: Shape index illustration for nine different surface categories, all with the same curvedness. Figure by [59].

### 3.3.4 Manifold Learning

The idea of using data not as isolated points but as belonging to a manifold has multiple applications in the machine learning area [51, 4]. In particular, we will be interested in the manifold learning area for dimensionality reduction [19] in order to estimate the curvature.

Intuitively, the learning of manifolds is interested in finding representations of points in a $n$-dimensional manifold in a new $n_0$-dimensional manifold, where $n_0 < n$, preserving some property of the original, such as the distance between points or neighborhood. The first strategies usually were in two stages: estimate the distances; use multidimensional, metric scaling [106, 44] or not [61, 62], to obtain a new representation.

Consider a continuous function $\mathbf{r} : [0, 1] \to \mathbb{R}^n$, such that $\mathbf{r}(0) = \mathbf{a}$ and $\mathbf{r}(1) = \mathbf{b}$, and a manifold associated to function $f$ with domain $\mathbb{R}^n$. the function $(f \circ \mathbf{r})(t)$ defines a trajectory over the manifold in study and its length $L$ is:

$$L = \int_0^1 \left\| \frac{d(f \circ \mathbf{r})(t)}{dt} \right\| dt. \tag{3.31}$$

the composition $(f \circ \mathbf{r})(t)$ such that $\mathbf{r}(0) = \mathbf{a}$ e $\mathbf{r}(1) = \mathbf{b}$ which minimizes the length $L$ is called the **geodesic** between points $\mathbf{a}$ and $\mathbf{b}$ on the manifold of $f$.

To find analytical solutions for the geodesics of arbitrary functions is not a simple problem. A common strategy is to estimate numerically with the use a piecewise linear trajectory with a finite number of segments. The coordinates of the points of the segments can be optimized through algorithms such as Newton-Raphson or Gradient Descendent [8]. The figure 3.8 exemplifies this method. However, these methods are expensive and impracticable when you want to estimate the geodesics for each pair of a cloud of points over the manifold, since they require multiple iterations to converge [8].

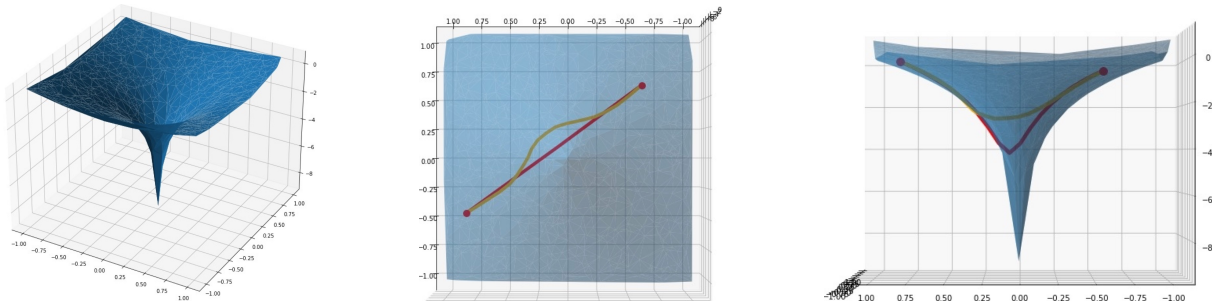

Figure 3.8: Example of a numerical estimate of a geodesic. In red we have the naive straight line between the points and in yellow is the optimized trajectory. (Left) Diagonal perspective. (Center) Top perspective. (Right) Side-view perspective.

Thus, it is necessary a new approach to estimate these geodesics. The first algorithm to do this was Isomap [105]. Its idea, illustrated in the figure 3.9 below, is to estimate the geodesics through the minimum distance in a graph of the nearest $k$ neighbors.

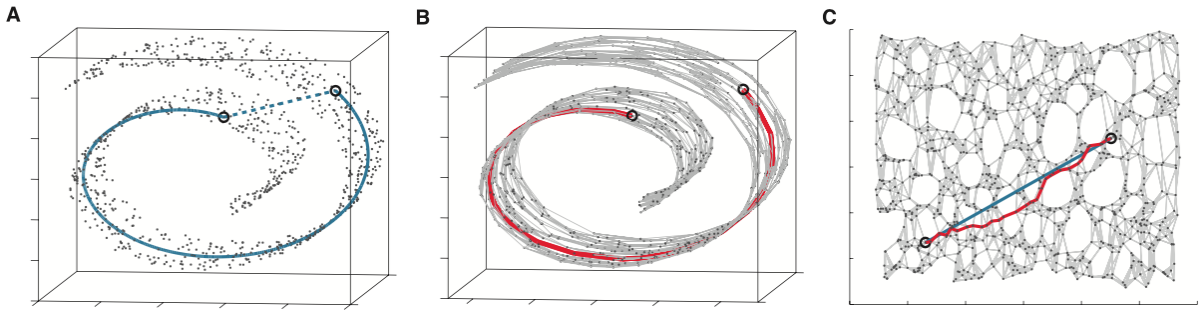


Figure 3.9: (**A**) Euclidean distance between two points do not represent the real distance on the implicit lower dimensional manifold. (**B**) Construct a graph with the $k$ nearest neighbors for each point. With said graph is possible to estimate the real geodesic. (**C**) Two dimensional representation which best preserves the shortest distances on the graph. Figure by [105].

The representation that best preserves the geodesics does not necessarily preserve other properties of the manifolds, especially the similarity in a neighborhood. In the previous image itself we can observe that Isomap creates regions, holes, which force the representation to have minimal distances similar to the original but that violate the genus of the original manifold. As our proposed curvature analysis has a step of interpolation, interpolating these regions may not have any sense.

The Locally Linear Embedding [94] has a different approach that eliminates the need to estimate the geodesics of very distant pair of points. The algorithm is based on simple geometric intuitions. If the manifold is sufficiently sampled, the neighborhood of a point is close to a locally linear region. We can find the matrix $W$ of linear coefficients that minimizes the reconstruction error of a point by its neighbors:

$$\varepsilon(W) = \sum_i \left| X_i - \sum_j W_{ij} X_j \right|^2, \tag{3.32}$$

in which $W_{ij} = 0$ if $X_j$ is not in the neighborhood of $X_i$ and $\sum_j W_{ij} = 1$. As in Isomap, we will define the neighborhood as the $k$ closest points. The optimal weights $W$ can be found with the minimum squares method. The reconstruction errors are invariant to rotations, rescaling, and translation of a point and its neighborhood. By this symmetry, it follows that the reconstruction weights characterize the intrinsic geometric properties of each neighborhood, and therefore will be the algorithm used in our analyses.

To obtain the manifold representation in a smaller dimension, one need only to obtain the coordinates $Y$ in this new dimension which minimize

$$\Phi(Y) = \sum_i \left| Y_i - \sum_j W_{ij} Y_j \right|^2. \tag{3.33}$$

Subject to the constraints that make the problem well behaved, the reconstruction error can be minimized by solving a sparse eigenvalue problem in the size of the number of samples.

### 3.3.5 Alpha-Shape Algorithm

To calculate how the gradient of the manifold learning coordinates behaves, it is necessary that we interpolate the values for a whole likely region, not explicitly defined. A possible approach is to use the convex hull [14] but a large region can represent impossible coordinates in the original domain, it is necessary a region that contains the points obtained but that is tighter. Thus, the algorithm $\alpha$-Shape [35] was used.

While convex hull is the smallest convex polygon that contains all the points of a set, we now have the notion of $\alpha$-shape, defined as follows:

(a) A point $p$ in a set $S$ is $\alpha$-extreme if exists a closed disk with radius $1/\alpha$ which contain all points of $S$ and $p$ is in its border;

(b) Two points $p$ and $q$ in $S$ are $\alpha$-neighbors if there exist a closed disk of radius $1/\alpha$ which contain all points of $S$ and $p$ and $q$ are on its border;

(c) The $\alpha$-shape is defined as the graph of straight lines in which the vertices are $\alpha$-extremes and the edges connect two $\alpha$-neighbors.

We can still extend the disk definition, in which a negative radius disk is the complement of a disk with a radius equal to its absolute value. Thus, a $\alpha$-hull may be non-convex and possess holes.

### 3.3.6 Mean Value Coordinates

For better analysis of the results we will see in the section 6.2 on the curvature of the future cost function, it will be beneficial to map an irregular polygon with an arbitrary number of $k$ of vertices into a regular polygon.

The Mean Value Coordinates are a generalization of the baricentric coordinates for polygons with more than 3 vertices using a planar triangulation expressed as a convex combination of the neighboring vertices. Unlike the triangle case, polygons with more than 3 vertices have infinite possible coordinate systems that satisfy

$$\sum_{i=1}^{k} \lambda_i v_i = u, \tag{3.34}$$

$$\sum_{i=1}^{k} \lambda_i = 1, \tag{3.35}$$

in which $u$ is a point inside the polygon, $v_i$ are the vertices and $\lambda_i$ are its coordinates. The advantage of this method over others is its low cost and the fact that it is a smooth parametrization in the vertices. So, we will not have discontinuities and the analysis can be done in the new polygon.

Be $0 < \alpha_i < \pi$ the angle in $u$ on the triangle $(u, v_{i-1}, v_i)$, defined in a cyclic way. Its coordinates are given by

$$\lambda_i = \frac{w_i}{\sum_{j=1}^{k} w_j}, \quad w_i = \frac{\tan(\alpha_{i-1}/2) + \tan(\alpha_i/2)}{\|v_i - u\|}. \tag{3.36}$$

## 3.4 Shapley Additive Explanations (SHAP)

When working with machine learning models, not always are clear the reasons that led them to return one output in relation to the other. In order to verify whether the models actually learn what they were proposed to investigate on a system, and not only spurious patterns, it is important to understand the reasons. There are models, such as decision trees, which naturally explain their reasons, but in those more complex and opaques, such as the neural models, this is not trivial. Thus began the search for the explainability and interpretability of models.

this distinction is not always explict, but [95] defines them as:

- **Interpretability**: development of inherently interpretable models;
- **Explainability**: attempt of explaining "black box" models in a *post hoc* way.

Neural networks are generally "black box" models of little interpretability. In this sense, the explainability techniques make more meaning, even if there are interpretable

neural models [114, 108]. A widely used explainability technique for neural networks is Shapley's Additive Explanations (SHAP) [68].

SHAP combines two previous approaches, LIME [91] and Shapley regression values [65]. A brief introduction of these will follow.

### 3.4.1 Local Interpretable Model Agnostic Explanations (LIME)

The Local Interpretable Model Agnostic Explanations (LIME) seek to explain the predictions of any classifier or regressor in a fair way by approximating it locally with an interpretable model. Here we have explanations of single predictions and not of the model's behavior as a whole.

For a given instance, disturbances are sampled in its neighborhood and its predictions are evaluated. An interpretable model, such as a sparse linear model, using Lasso or Ridge, is trained on that set of disturbed samples and its predictions. From the interpretation of this local model, we can get the local behavior of our original model. It reminds the argument used with LLE in the manifold learning problem.

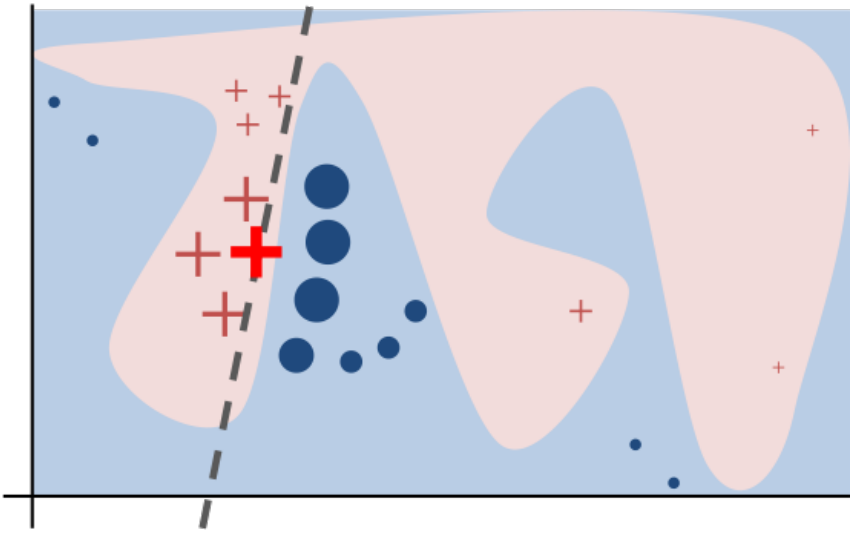

Figure 3.10: Synthetic intuitive example for LIME. Figure by [91].

Consider a decision function $f$ unknown by LIME, represented by the blue and pink regions in the figure 3.10, which is not well modeled as a linear function. In the image, the bold red cross represents the point to be explained. LIME samples points in the domain of $f$ and defines weights accordingly to the proximity to the point of study. Explanations are a linear function that explains local behavior faithfully but not globally.

The strategy to understand a model becomes how to obtain representative instances of the entire domain of application, since we can study the local explanations of each one.

### 3.4.2 Shapley Regression Values

In [65], we have a sensitivity analysis method to study the importance of a model input. Here we want to consider the interactions and redundancies between the different entries of an instance. The method considers the effect of each feature in combination with the others by disturbing all subset of features and observing how this affects the model prediction.

The method first trains a model on a set of data. Then for each instance, the method disturbs all the input subset and calculates the individual contribution of each:

$$\phi_i(x) = \sum_{Q \subseteq S \setminus \{i\}} \frac{|Q|!(|S| - |Q| - 1)!}{|S|!}(\Delta_{Q \cup \{i\}}(x) - \Delta_Q(x)), \tag{3.37}$$

in which $S = \{1, 2, ..., n\}$ is the input set, $\Delta_Q(x) = f_Q(x) - f_{\{\}}(x)$ is the gain of set $Q$ and $f_Q(x) = \mathbb{E}[f|X_i = x_i, \forall i \in Q]$ is the expected value of the model when only using the inputs in $Q$.

Since equation 3.37 has an exponential cost to be computed, the rest of the article is dedicated to justifying the approximate method to compute the contribution of each feature.

The Shapley values satisfy:

(a) Local Accuracy: the explanations must be representative of a small region in the surroundings of the instance;

(b) Absence: features with irrelevant contributions shouldn't affect the explanations;

(c) Consistency: the explanations must be coherent between different models.

### 3.4.3 Shapley Additive Explanations

Shapley Additive Explanations (SHAP) [68] are a method that combines the advantages of LIME and Shapley values to get a more faithful interpretation of the predictions

of a model. As discussed, LIME uses a linear explanation to approach a function $f$ in a small region in the instance sorroudings; and Shapley values are additive attributes that satisfy local accuracy, absence and consistency. This means that they attribute a value of importance to each feature individually for their contribution to the prediction.

SHAP uses Shapley values as instance weights in the linear regression of LIME. Thus, non-linear interactions between features are considered. The authors go beyond and present theoretical guarantees in accuracy and consistency.

# Chapter 4

# Modelling

In this chapter we have the mathematical definitions for the Control Problem in One Dimension, the Hidrothermal Dispatch Problem and the Optimal Power Flow problem.

## 4.1 Control Problem in One Dimension

The Control Problem in One Dimension is a pretty simple problem, which makes it possible to get the actual future cost functions. It is defined by:

$$\begin{aligned} \min_{\mathbf{x}\in\mathbb{U}} \quad & \mathbb{E}\left[\sum_{t=1}^{T} \beta^{t-1}|x_t|\right] \\ & x_t = x_{t-1} + c_t + \xi_t \\ & c_t \in \{\pm 1\}. \end{aligned} \tag{4.1}$$

The state $x_t$, the control variable $c_t$ and the uncertainty $\xi_t$ are one-dimensional. The goal is to minimize the expectation of the distance between $x_t$ and the origin, subject to the decay factor of $\beta$ throughout the $T$ planning period. In this work, it was used $T = 8$, $\beta = 0.9$, $x_0 = 2$, and the uncertainty of $\xi_t$ was symmetrically sampled around zero, as in [3]. As a simple problem, we can pay the exponential cost and calculate the actual future cost functions for a small sample.

## 4.2 Hidrothermal Dispatch Problem

The objective of the Hidrothermal Dispatch Problem is to determine a operation strategy, for each stage, given the current state of the environment, for how much each

plant (thermal or hydrothermal) should generate of energy. This strategy should minimize the average operating cost, which consists in the cost of the thermal fuel and for a penalty in case the demand is not supplied. The availability of limited amounts of hydrothermal energy, in the form of reserve water, makes the problem quite complex. If the reservoirs are exhausted and small rainwater flows occur, it may be necessary to use very expensive thermal to meet the demand. On the other hand, if the reservoirs remain at high levels and intense inflows occur, we may have spilled water, i.e. wasted energy.

### 4.2.1 Conventional Formulation

In this formulation of the problem, the state variables $x$ represent the water volume of the reservoirs; the decision variables $u$ are how much to turbine $q$ and spill $s$ from each reservoirs, how much to generate from each thermal plant $g$, and how much $d$ of energy deficit we will have in the system; and the random variables $\omega$ are the rain inflow of each reservoirs.

Be $p$ the cost associated with the penalty of energy deficit and $c$ the costs for the fuel of the thermal power plants, the cost $C_i$ of the subproblem for stage $i$ is simply

$$C_i(x, u, \omega) = p \cdot d + \sum_{j \in \mathcal{T}} c_j \cdot g_j, \tag{4.2}$$

constrained by

$$d + \sum_{k \in \mathcal{H}} e_k \cdot q_k + \sum_{j \in \mathcal{T}} g_j = l_i, \tag{4.3}$$

$$g_j \leq \bar{g}_j \quad \forall j \in \mathcal{T}, \tag{4.4}$$

$$q_k \leq \bar{q}_k \quad \forall k \in \mathcal{H}, \tag{4.5}$$

in which $\mathcal{T}$ is the set of thermal plants, $\mathcal{H}$ is the one of hydro plants, $e$ is the exchange factor of water to energy, $l$ is the demand, $\bar{g}$ is the maximum thermal generation and $\bar{q}$ is the maximum turbinated outflow of the reservoir.

Let $\mathcal{M}(i)$ be the set of hydroelectric plants above $i$, the transition function $T_i(x, u, \omega)$ is given by:

$$x'_j = T_i(x_j, u_j, \omega_j) = x_j - q_j - s_j + \omega_j + \sum_{k \in \mathcal{M}(j)} q_k + s_k \quad \forall j \in \mathcal{H}, \tag{4.6}$$

in which the variable $x'$ is limited by:

$$x'_j \leq \bar{x}_j \quad \forall j \in \mathcal{H}. \tag{4.7}$$

### 4.2.2 Extensions

The problem described above and the target of our work can be extended to better represent a real scenario, without the need to add non-linear constraints or costs. For example, in many locations the water of the reservoirs plays other functions for society, such as fish production and tourism, and therefore may have a minimum required volume.

In a more complex way, the Brazilian electric system is divided into four subsystems. Each of them has a demand and its own plants and reservoirs, but it is allowed to transfer part of the energy from one subsystem to another in order to meet the different demands. Here the subsystems have to act together and are not completely independent.

There is also the possibility of adding a water conversion to nonlinear energy, with a piecewise linear approximation [99]. The idea is that the exchange factor in real reservoirs depends on its water levels and is not constant.

## 4.3 Optimal Power Flow

In the problem of hydrothermal dispatch it is assumed that the energy generated by any power plant is easily accessible to all users of the electric grid. The reality is very different. The grid is highly complex and has various constraints regarding the distribution of the energy generated at a given point. The problem of the optimal power flow [17], differently, relates to the distribution of the energy generated, in which the plants that generate and the customers that demand are in a graph. If in the dispatch we abstracted the distribution, here we abstract the specific notions of the generation, such as the reservoirs of the hydroelectric plants. For a real application, it is of interest to optimize the problem that unites the previous two.

Given an electric grid (with physical restrictions), a set of generators and a set of consumers, we want to minimize the generation of energy needed to satisfy the demand of consumers. What makes the multi-stage problem is the incorporation of energy storage systems. In its traditional formulation, it is a subproblem of non-convex optimization

with both quadratic constraints and stage objectives.

### 4.3.1 Constants

When dealing with physical artefacts related to electric circuits, it is only natural to use complex numbers. We will use $j$ to denote the square root of $-1$; for a complex variable $z$, $\mathrm{Re}(z)$ and $\mathrm{Im}(z)$ denote the real and imaginary part of $z$; $|z|$ is the magnitude and $z^*$ denote the conjugate. For a square matrix $A$, $\mathrm{Tr}(A)$ denote the trace, i.e. the sum of the values of the diagonal.

Consider the graph$\mathcal{G} = (\mathcal{N}, \mathcal{E})$ for the electric network, in which the vertices $i \in \mathcal{N}, j \in \mathcal{N}$ represent buses and the edges $\mathcal{E}$ are branches that connect a pair $(i, j)$ of buses. $\mathcal{N}^G \subseteq \mathcal{N}$ is the set of buses with access to generators. Set $L$ has edges in $\mathcal{E}$ but in both directions, i.e. $(i, j) \in \mathcal{E} \implies (i, j), (j, i) \in \mathcal{L}$. For a node $i$, $k \sim i$ denotes that node $k$ is connected to $i$.

Transmission lines lose power due to resistance ($r_{ij}$), which measures the opposition to current flow. For AC circuits, there is the reactance effect ($X_{ij}$), which stores and releases power to the circuit in a delayed way. The two quantities can be quantified in one, the impedance: $z_{ij} = r_{ij} + \mathrm{j}x_{ij}$. It will be of our interest its inverse, the admittance, which in turn can be decomposed into two values, the conductance and the susceptance, $y_{ij} = 1/z_{ij} = g_{ij} + \mathrm{j}b_{ij}$. If $i \not\sim j$, then we define $y_{ij} = 0$.

The admittance matrix, which relates the current injection and bus power, is defined as

$$Y_{\text{bus}} = \begin{cases} -y_{ij}, & i \neq j \\ y_{ii} + \sum_{k \sim i} y_{ik} & i = j \end{cases}, \tag{4.8}$$

where $y_{ii}$ is the admittance to ground at bus $i$. let $\zeta_i$ be a vector of length $|\mathcal{N}|$ with 1 on the $i$-th entry and zero otherwise:

$$Y_{\text{bus},i} := \zeta_i \zeta_i^T Y_{\text{bus}} \tag{4.9}$$

$$Y_{\text{bus},ij} := y_{ij} \zeta_i \zeta_i^T - y_{ij} \zeta_i \zeta_j^T \tag{4.10}$$

$$Y_i := \frac{1}{2} \begin{bmatrix} \text{Re}(Y_{\text{bus},i} + Y_{\text{bus},i}^T) & \text{Im}(Y_{\text{bus},i}^T - Y_{\text{bus},i}) \\ \text{Im}(Y_{\text{bus},i}^T - Y_{\text{bus},i}) & \text{Re}(Y_{\text{bus},i} + Y_{\text{bus},i}^T) \end{bmatrix} \tag{4.11}$$

$$\bar{Y}_i := -\frac{1}{2} \begin{bmatrix} \text{Im}(Y_{\text{bus},i} + Y_{\text{bus},i}^T) & \text{Re}(Y_{\text{bus},i} - Y_{\text{bus},i}^T) \\ \text{Re}(Y_{\text{bus},i} - Y_{\text{bus},i}^T) & \text{Im}(Y_{\text{bus},i} + Y_{\text{bus},i}^T) \end{bmatrix} \tag{4.12}$$

$$Y_{ij} := \frac{1}{2} \begin{bmatrix} \text{Re}(Y_{\text{bus},ij} + Y_{\text{bus},ij}^T) & \text{Im}(Y_{\text{bus},ij}^T - Y_{\text{bus},ij}) \\ \text{Im}(Y_{\text{bus},ij}^T - Y_{\text{bus},ij}) & \text{Re}(Y_{\text{bus},ij} + Y_{\text{bus},ij}^T) \end{bmatrix} \tag{4.13}$$

$$\bar{Y}_{ij} := -\frac{1}{2} \begin{bmatrix} \text{Im}(Y_{\text{bus},ij} + Y_{\text{bus},ij}^T) & \text{Re}(Y_{\text{bus},ij} - Y_{\text{bus},ij}^T) \\ \text{Re}(Y_{\text{bus},ij} - Y_{\text{bus},ij}^T) & \text{Im}(Y_{\text{bus},ij} + Y_{\text{bus},ij}^T) \end{bmatrix} \tag{4.14}$$

$$M_i := \begin{bmatrix} \zeta_i \zeta_i^T & 0 \\ 0 & \zeta_i \zeta_i^T \end{bmatrix} \tag{4.15}$$

$$M_{ij} := \begin{bmatrix} (\zeta_i - \zeta_j)(\zeta_i - \zeta_j)^T & 0 \\ 0 & (\zeta_i - \zeta_j)(\zeta_i - \zeta_j)^T \end{bmatrix} \tag{4.16}$$

Theses constants will be important to guarantee the physical laws that rule circuits behaviors [58, 80].

### 4.3.2 Original Formulation

In this problem formulation, based on the work of [43], the state variables $x$ represent the stored energy $b_i$ and the energy generation $p_i^G$ and $q_i^G$ (real and imaginary part); the decision variables $u$ are the charge/discharge rates $r_i$ of storage, the power $v_i$ and the flows of $p_{ij}$ and $q_{ij}$ (real and imaginary part); and the random variables $\omega$ are the demands of $p_i^D$ and $q_i^D$ (real and imaginary part). As $v_i$ is complex, we'll use $\mathcal{V} = \begin{bmatrix} \text{Re}(v)^T & \text{Im}(v)^T \end{bmatrix}^T$. The reader might wonder why modelling the generation as a state variable. The reason is the difference constraint, inequations 4.27 and 4.28, which require the value of the previous stage (the same trick used for autoregressive modeling discussed in the 2.1.3 subsection).

Let $c_i^{(k)} \geq 0$ be the cost of order $k$ associated to generation at $i$, the subproblem cost $C_i$ for stage $i$ is simply

$$C_i(x, u, \omega) = \sum_{i \in \mathcal{N}} c_i^{(2)} (p_i^{G\prime})^2 + c_i^{(1)} p_i^{G\prime} + c_i^{(0)}, \tag{4.17}$$

i.e. a convex quadratic function. Other costs may be chosen, like the minimum thermal loss in transmission. The constraints for the equilibria of real and reactive power for each bus is:

$$p_i^{G\prime} - p_i^D - r_i = \mathcal{V}^T Y_i \mathcal{V} \qquad i \in \mathcal{N} \tag{4.18}$$
$$q_i^{G\prime} - q_i^D = \mathcal{V}^T \bar{Y}_i \mathcal{V} \qquad i \in \mathcal{N} \tag{4.19}$$

Then, we have the constraints the govern the flow, including its magnitude (quadratic)

$$p_{ij} = \mathcal{V}^T Y_{ij} \mathcal{V} \qquad (i,j) \in \mathcal{L} \tag{4.20}$$
$$q_{ij} = \mathcal{V}^T \bar{Y}_{ij} \mathcal{V} \qquad (i,j) \in \mathcal{L} \tag{4.21}$$
$$q_{ij}^2 + q_{ij}^2 \leq (S_{ij}^{\max})^2 \qquad (i,j) \in \mathcal{L} \tag{4.22}$$

The constraints that bound the energy generation

$$P_i^{\min} \leq p_i^{G\prime} \leq P_i^{\max} \qquad i \in \mathcal{N}^G \tag{4.23}$$
$$Q_i^{\min} \leq q_i^{G\prime} \leq Q_i^{\max} \qquad i \in \mathcal{N}^G \tag{4.24}$$

The constraints below bound the power magnitude and thermal loss

$$(V_i^{\min})^2 \leq \mathcal{V}^T M_i \mathcal{V} \leq (V_i^{\max})^2 \qquad i \in \mathcal{N} \tag{4.25}$$
$$\mathcal{V}^T M_{ij} \mathcal{V} \leq L_{ij}^{\max} \qquad (i,j) \in \mathcal{L} \tag{4.26}$$

Difference constraints,

$$\Delta P_i^{\min} \leq p_i^{G\prime} - p_i^G \leq \Delta P_i^{\max} \qquad i \in \mathcal{N}^G \tag{4.27}$$
$$\Delta Q_i^{\min} \leq q_i^{G\prime} - q_i^G \leq \Delta Q_i^{\max} \qquad i \in \mathcal{N}^G \tag{4.28}$$

And finally, the constraints that rule the storage system, the transition function (in addition to the generation levels)

$$B^{\min} \leq b_i' \leq B^{\max} \qquad i \in \mathcal{N}^S \tag{4.29}$$
$$R^{\min} \leq r_i \leq R^{\max} \qquad i \in \mathcal{N}^S \tag{4.30}$$
$$b_i' = b_i + \eta r_i \qquad i \in \mathcal{N}^S \tag{4.31}$$

in which $\eta$ is the battery efficiency.

### 4.3.3 Penalization for Temporal Consistency

The constraints at inequalities 4.27 and 4.28 impose a temporal relationship. The current stage solution is not only affected by the future through the future cost function, but also by the need for the future stages to have viable solutions. This is difficult to guarantee always, even more during the beginning of training in which the future cost function can lead the model to extreme regions of the state space. Therefore, in this work we used a different modeling, in which the effective difference between stages (difference past a certain level) is penalized in a quadratic form through the following inequalities instead of 4.27 and 4.28

$$-\Delta P_i \leq p_i^{G\prime} - p_i^G \leq \Delta P_i \qquad i \in \mathcal{N}^G \tag{4.32}$$

$$\Delta P_i - \Delta P_i^{\max} \leq \Delta P_i^E \qquad i \in \mathcal{N}^G \tag{4.33}$$

$$-\Delta Q_i \leq q_i^{G\prime} - q_i^G \leq \Delta Q_i \qquad i \in \mathcal{N}^G \tag{4.34}$$

$$\Delta Q_i - \Delta Q_i^{\max} \leq \Delta Q_i^E \qquad i \in \mathcal{N}^G \tag{4.35}$$

$$\Delta P_i^E, \Delta Q_i^E \geq 0 \qquad i \in \mathcal{N}^G \tag{4.36}$$

and the addition of quadratic components in the objective function

$$\begin{aligned} C_i(x,u,\omega) = &\sum_{i\in\mathcal{N}^G} c_i^{(2)}(p_i^{G\prime})^2 + c_i^{(1)} p_i^{G\prime} + c_i^{(0)} \\ &+ \sum_{i\in\mathcal{N}^G} a_i^{(2)}(\Delta P_i^E)^2 + a_i^{(1)}\Delta P_i^E \\ &+ \sum_{i\in\mathcal{N}^G} b_i^{(2)}(\Delta Q_i^E)^2 + b_i^{(1)}\Delta Q_i^E \end{aligned} \tag{4.37}$$

# Chapter 5

# Future Cost Estimator

In this chapter we describe our algorithms for the Stochastic Neural Dynamic Programming (SNDP) algorithm, with some additional discussion over implementation details.

## 5.1 Algorithm

The algorithm is based on the original SDDP, in which is kept the sampling of states with a forward pass of the algorithm and the adjustment of the model with a backward pass. What differs is that now the adjustment is done on a neural network. Thus, it is possible to condition the future cost function with the observations of uncertainty, add memory of the previous stages and even use external and non-structured data. In a context of farm decision-making, for example, a satellite image of your farm can report more than one-dimensional statistics such as rainfall.

The goal of the algorithm is to create a projection of the future cost manifold into the set of manifolds that can be represented as integer programs, which we know how to optimize (even though not in a polynomial way). It can therefore be thought that we are looking for something similar to an algorithm of reinforcement learning, as described in the subsection 2.2.1, only with single network. Instead of learning a distribution of actions around the optimum, we use optimization to always choose the optimum.

We will describe each step individually, but its basic structure is as following:

- **Forward Pass**: we simulate the policy stage by stage, from the beginning to the end. At each stage, a realization of the random variable is observed and the approximate subproblem is solved to generate a candidate state for the next stage.
- **Backward Pass**: visit the stages in inverse order, so that by refining a stage, we will already have refined its successor.

We sample the observations at random, but one can make the modification to

always sample in order, like a mini-batch of regular neural network training. All samples appear the same amount of times. Algorithmically, we have both approaches below:

**Algorithm 3:** SNDP — Sampling

**Data:** Number of iterations $n$, number of simulated samples $f$, number of random samples over the domain $d$, set $S$ of scenarios and model $G$ to be trained.
**Result:** Policy for the problem.
**for** $i = 1, ..., n$ **do**
  States $\leftarrow$ Forward-Pass$(G, S, f) \cup$ Sample-over-Domain$(d)$;
  $G \leftarrow$ Backward-Pass(States, $G, S$);
**end**

**Algorithm 4:** SNDP — *Mini-Batch*

**Data:** Number of epochs $n$, batch size $b$, number of random samples over the domain $d$, set $S$ of scenarios and model $G$ to be trained.
**Result:** Policy for the problem.
**for** $i = 1, ..., n$ **do**
  **for** $j = 1, ..., |S|/b$ **do**
    States $\leftarrow$ Forward-Pass$(G, S, b, j) \cup$ Sample-over-Domain$(d)$;
    $G \leftarrow$ Backward-Pass(States, $G, S$);
  **end**
**end**

The forward pass function in the algorithm 4 receives the indices of the scenarios that will be used, instead of how many scenarios to sample. This function only simulates the policy for each of the scenarios and therefore will be considered as already explained. The backward pass function will be explained in the following subsections, since it is in it that we have a few modifications.

The architecture of the neural network that represents the future cost function (which we will call the network to the right) is independent of the network (or rather hyper-network) architecture that proposes the weights (the network to the left). If we increase the complexity of the network to the left, the cost of solving the subproblem is not affected. However, we have to pay attention to the cost of solving the optimization of the network to the right. This value is not trivial to know and is usual to use other values as heuristic, such as the number of binary variables and the number of constraints, given in the table 5.1.

On SDDP we have the convergence guarantee for problems with i.i.d. uncertainty and linear constraints and objective function; by adding binary variables, SDDiP guarantees the convergence to the optimum. The algorithm proposed in this work does not offer any theoretical proof that justify its convergence. However, some intuitions are given

| Type of manifold | Binary variables | Constraints |
|---|---|---|
| Convex | 0 | # Cuts |
| $k$-Convex | $k$ | $k \cdot (\#\text{ Cuts}) + 2$ |
| ReLU Network | # Neurons | $3 \cdot (\#\text{ Neurons}) - 1$ |

Table 5.1: Comparison between the integer programming formulations for the different kinds of manifolds.

below that seek to justify its empirical convergence in experiments. Following the idea of learning propagation, it can be argumented inductively:

- **Base**: The penultimate stage has training instances $\{X, Y\}$ that represent your real future cost. If the network has enough capacity, with a finite number of iterations it will converge to the surroundings of the real future cost function.
- **Inductive step**: If the $k$ stage converged to surroundings of its real future cost function, then the $k-1$ stage may have training instances $\{X, Y\}$ that represent its real future cost. If the network has sufficient capacity and the **training does not interfere with subsequent stages**, with a final number of iterations it will converge to its real future cost function environment.

It remains to know for which problems a given network to the right has the ability to converge to a given error tolerance and whether the proposed algorithms guarantee that an earlier stage does not disturb much its subsequent stages.

### 5.1.1 Convergence to the Mean

In this formulation, we want, as in the original SDDP, to estimate the average future cost of a given state in the current stage. Unlike SDDP, in which each stage has an independent set of cuts, the algorithm has a single neural network (net to the left). Its input is a binary representation of the stage and its output is the weights of the future cost function (net to the right): a set of cuts, a set of $k$-cuts or the weights of a ReLU Neural network. These functions have different possible representations (convex, $k$-convex and not convex) and number of binary variables (an indication of potential difficulty to solve). The knowledge of a specialist can be to the help of which manifold type is the best output.

As seen earlier, both the $k$-convex manifolds and the ReLU networks can be incorporated into the subproblems and solved by integer programming solvers. Thus, the forward pass does not need any change and is kept as described before. However, the

**Algorithm 5:** Backward Pass — Convergence to the mean

**Data:** States $E$, set $S$ of scenarios and model $G$ to be trained
**Result:** Future Cost Function Update
$X \leftarrow \varnothing$;
$Y \leftarrow \varnothing$;
**for** $t = T, ..., 2$ **do**
  **for** $s = 1, ..., S$ **do**
    cost $\leftarrow 0$;
    **for** $r = 1, ..., |S|$ **do**
      Solve the problem for stage $t$ with initial state $E_{t,s}$ and observation $\Omega_{t,r}$;
      cost $\leftarrow$ cost + optimal solution cost;
    **end**
    $X \leftarrow X \cup \{E_{t,s}\}$;
    $Y \leftarrow Y \cup \{\text{cost}/S\}$ ;
  **end**
  $\hat{Y} \leftarrow G(\text{Stage Indices}, X)$;
  loss $\leftarrow$ Loss-Function$(Y, \hat{Y})$;
  Do one step of the optimization algorithm (e.g. Gradient descent);
**end**

backward pass must be changed, as can be seen in the algorithm 5. The optimization algorithm used was Adam [57].

Be $\sigma$ the expected computational cost of solving a subproblem. We know that solving integer programs is exponential in the worst case, and depends on the constraints of each problem. So we don't have a way to calculate its cost agnostically to the problem or architecture of the chosen network. The cost of this bacward pass is therefore $O(\sigma T S^2)$. See that unlike SDDP, we have multiple simulated and optimized scenarios in parallel. As we will see in the sub-section 5.2.1 about the loss function, it's essential to have more than one scenario per iteration.

The idea of the algorithm is to reinforce the information of the latter stages (which appear first in the backward pass) compared the first ones, since the most reliable information about the future is the closest to the end. Learning is then propagated to the earliest stages, without allowing the network to deviate too much of the ideal for the final stages.

A possible backward pass modification is to update the weights only once in the backward pass, after having calculated the estimated future cost of each stage. This modification will be investigated in the 7.2.1 sub-section of the experiments.

### 5.1.2 Conditioned on the Uncertainty

We know that the network to the left does not affect the complexity of the subproblem to the right. Thus, as long as we manage to modify the algorithm 5 to incorporate new data, we can use even more information and distinct and more complex architectures for the network to the left. In this sub section, we will discuss the modification that allows to incorporate the uncertainty conditioning to the network to the left. The algorithm inference scheme is described in the figure 5.1.

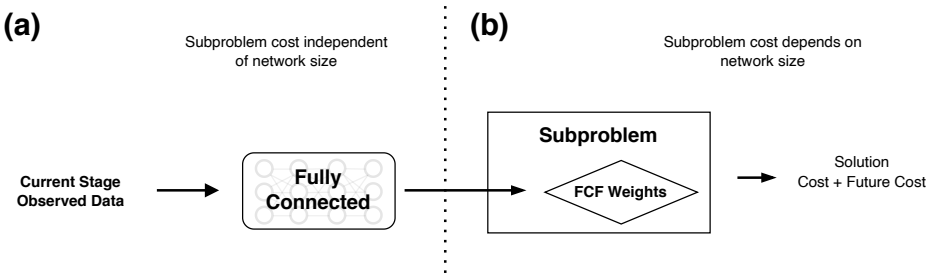


Figure 5.1: Conditional SNDP schema.

As uncertainty possesses temporal dependence, knowing the realization for this stage informs us about the scenario as a whole, both about the past decisions as well as the decisions that will still have to be taken. Thus, in the backward step we will no longer be able to sample independently to each stage, but rather to sample an entire scenario and simulate the policy on it. The great change is over the backward pass, described in the algorithm 6 below:

**Algorithm 6:** Backward Pass — Conditioned on the Uncertainty

**Data:** States $E$, set $S$ of scenarios e modelo a ser treinado $G$
**Result:** Future Cost Function Update
$X \leftarrow \varnothing$;
$Y \leftarrow \varnothing$;
**for** $t = T, ..., 2$ **do**
  **for** $s = 1, ..., S$ **do**
    Solve the problem for stage $t$ with initial state $E_{t,s}$ and observation $\Omega_{t,s}$;
    cost $\leftarrow$ optimal solution cost;
    $X \leftarrow X \cup \{E_{t,s}\}$;
    $Y \leftarrow Y \cup \{\text{cost}\}$ ;
  **end**
  $\hat{Y} \leftarrow G(\text{Stage Indices}, \text{Observations}, X)$;
  loss $\leftarrow$ Loss-Function$(Y, \hat{Y})$;
  Do one step of the optimization algorithm;
**end**

The main difference is therefore in the calculation of the future cost of the sampled state. We are no longer interested in the average of that value, but the value that would be obtained when we simulate the "future" of that observation. The goal is to optimize the average conditioned on the observation. Note that if each stage is an i.i.d. variable, the two expectations are equal. Thus, the algorithm acts on an even greater set of possible problems, those that have temporal dependence. Nothing prevents us from adding even more data, such as external information, predictions on future observations, the current state etc.

This approach has another advantage. Solving the subproblems is the most expensive step of an iteration. In the initial version, the algorithm 5, and also in SDDP, we calculate the future cost of a sample state for every possible realization, that is, for every future scenario. But that is no longer the case, we only compute the cost for a single future observation. Thus, the cost of a backward pass is $O(\sigma TS)$, scaling linearly in the number of scenarios.

### 5.1.3 Incorporating Memory

The previous approach is interesting because it conditions the future cost function on an observation, thus portraying much better the real future cost of that scenario. So it would be interesting if the function was conditioned not only in the current observation but also in all the previous observations. A possible way of doing that, portrayed in the figure 5.2, is the use of a memory vector and a LSTM network. In practice, it is the same previous algorithm only that conditioned in a memory vector instead of an observation. The modified version of the backward step is described in the algorithm 7.

This modification brings a difficulty. As a backward pass of the algorithm changes the network multiple times, the hidden state cannot be sampled along with the state during the forward pass. Its representation changes. At each modification of the network we have to pass again the observations by the LSTM cell in order to have an instance of the hidden state matching with the current weights of the network.

**Algorithm 7:** Backward Pass — With Memory

**Data:** States $E$, set $S$ of scenarios and the model $G$ to be trained
**Result:** Future Cost Function Update
$X \leftarrow \varnothing$;
$Y \leftarrow \varnothing$;
**for** $t = T, ..., 2$ **do**
  **for** $j = 1, ..., T$ **do**
    Compute the hidden state for scenarios in $S$.
  **end**
  **for** $s = 1, ..., S$ **do**
    Solve the problem for stage $t$ with initial state $E_{t,s}$, observation $\Omega_{t,s}$ e corresponding hidden state;
    cost $\leftarrow$ optimal solution cots;
    $X \leftarrow X \cup \{E_{t,s}\}$;
    $Y \leftarrow Y \cup \{\text{cost}\}$ ;
  **end**
  $\hat{Y} \leftarrow G(\text{Stage Indices}, \text{Observations}, \text{Hidden State}, X)$;
  loss $\leftarrow$ Loss-Function$(Y, \hat{Y})$;
  Do one step of the optimization algorithm;
**end**

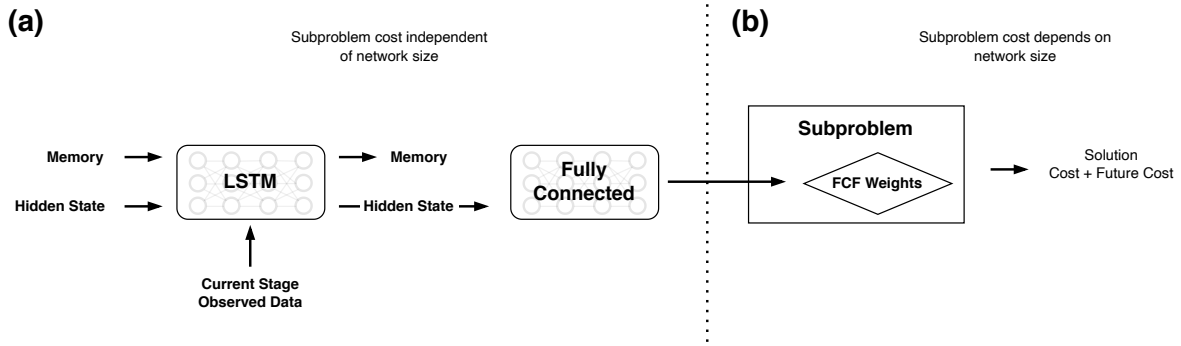


Figure 5.2: Memory SNDP schema.

## 5.2 Implementation Details

In this section we describe some modeling choices taken during the experimentation process that led to a good convergence of the algorithm. These details are the result of an empirical process and do not have formal arguments. Intuitions of possible reasons for their effectiveness are presented for each modification.

### 5.2.1 Loss Function

In the backward algorithms discussed in the previous section, the section 5.1, the weights are updated according to the loss of multiple stages, simultaneously. As the initial stages values involve a longer future, their future cost tends to be greater than that of the following stages. Depending on the scale of these values, the most initial and the most final stages have very different scales. If the $T$ stages have a similar current cost, the first future cost is close to $T$ times greater than the last. When calculating the loss function, the values relating to the initial stages will dominate and thus the signal of the final stages becomes very weak and the algorithm has difficulty to converge.

Considering the intuition of the propagation of learning, as the algorithm does not fix the cost of the final stages, since the firts ones dominate, learning is not longer propagated to the initials. For problems of small temporal window, or even of sparse costs or of small order, it is possible that losses in different stages have a similar order of magnitude and this issue is not a problem. But in general, the use of conventional losses functions, such as average square error or average absolute error, does not lead convergence to a good policy.

To solve this problem it is necessary to use a loss function invariant to the magnitude of the predictions. A traditional loss function for the task is the average absolute percentage error (MAPE), equation 5.1, in which the difference is normalized by the actual value, thus rescaling the values.

$$\text{MAPE} = \frac{1}{N}\sum_{i=1}^{N}\left|\frac{A_i - F_i}{A_i}\right|. \tag{5.1}$$

However, the MAPE function has its problems. If the costs assume values close to zero, the MAPE responses are extremely high, dominating over other errors. In addition, MAPE was criticized for penalizing more positive errors than the negative [71], it is not a symmetrical metric. Thus, [56] propose a modification to MAPE, to use the angle of inclination, equation 5.2. With this detail, the values close to zero are much better behaved (because it is an angle, the function is limited above and below, from 0 to $\pi/2$) and even though it is still asymmetric, MAPE is better distributed.

$$\text{MAAPE} = \frac{1}{N}\sum_{i=1}^{N}\arctan\left(\left|\frac{A_i - F_i}{A_i}\right|\right). \tag{5.2}$$

The attentive reader should be wondering about the elephant in the room. It is possible that a stage, especially the closest to the end of the planning horizon, has future cost zero. Both previous loss functions are not defined at that point. For problems that the future cost is almost never exactly zero, MAAPE works well. Therefore, this is not

really your problem. But, you may encounter another problem. As we are training a network, we need that the gradients related to the errors lead to a fast evolution of the network. However, the values to be predicted can take arbitrary magnitudes. MAAPE's derivative quickly approaches zero as the future costs increase, and thus the network practically does not evolve.

Ideally we want a loss function that satisfy, in simple terms:

(a) invariant to stage scale;

(b) well defined around zero and its neighborhood;

(c) approximatly symmetric;

(d) derivative which does not go to zero too fast.

A function that satisfy all above is given by equation 5.3, qhich we nicked the Mean Staged Squared Error (MSSE):

$$\begin{aligned} \text{MSSE} &= \frac{1}{NT}\sum_{i=1}^{NT}\left(\frac{A_i - F_i}{\sigma_{t(i)}}\right)^2 \\ &= \frac{1}{T}\sum_{t=1}^{T}\frac{1}{\sigma_t^2}\frac{1}{N}\sum_{i=(t-1)N+1}^{tN}(A_i - F_i)^2 \\ &= \frac{1}{T}\sum_{t=1}^{T}\frac{\text{MSE}_t}{\sigma_t^2} \end{aligned} \tag{5.3}$$

where $\sigma_{t(i)}$ is the standard deviation of both the predictions and target values of the stage the sample $i$ is part of. Its derivative is obtained below:

$$\begin{aligned} \frac{\partial \sigma_{t(i)}^2}{\partial F_i} &= \frac{2(F_i - \mu)(1 - \frac{\partial \mu}{\partial F_i})}{2N} + \sum_{j \neq i, \sigma_{t(j)} = \sigma_{t(i)}} \frac{2(\mu - c_j)\frac{\partial \mu}{\partial F_i}}{2N} \\ &= \frac{(F_i - \mu)(1 - \frac{1}{2N})}{N} + \sum_{j \neq i, \sigma_{t(j)} = \sigma_{t(i)}} \frac{(\mu - c_j)}{2N^2} \\ &= \frac{(F_i - \mu)(2N - 1)}{2N^2} + \frac{(2N-1)\mu - (2N\mu - F_i)}{2N^2} \\ &= \frac{(F_i - \mu)}{N} \end{aligned} \tag{5.4}$$

$$\begin{aligned}
\frac{\partial MSSE}{\partial F_i} &= \frac{1}{NT}\left(\frac{\partial}{\partial F_i}\frac{(A_i - F_i)^2}{\sigma^2_{t(i)}} + \sum_{j\neq i, \sigma_{t(j)}=\sigma_{t(i)}} \frac{\partial}{\partial F_i}\frac{(A_j - F_j)^2}{\sigma^2_{t(i)}}\right) \\
&= \frac{1}{NT}\left(\frac{(A_i - F_i)\left(-2\sigma^2_{t(i)} - (A_i - F_i)\frac{\partial \sigma^2_{t(i)}}{\partial F_i}\right)}{\sigma^4_{t(i)}} + \frac{1}{N}\frac{\partial}{\partial F_i}\frac{1}{\sigma^2_{t(i)}}\right) \\
&= \frac{1}{NT}\left(\frac{(A_i - F_i)\left(-2\sigma^2_{t(i)} - (A_i - F_i)\frac{\partial \sigma^2_{t(i)}}{\partial F_i}\right)}{\sigma^4_{t(i)}} - \frac{\frac{\partial \sigma^2_{t(i)}}{\partial F_i}}{N\sigma^4_{t(i)}}\right) \\
&= \frac{1}{NT}\left(\frac{(A_i - F_i)\left(-2\sigma^2_{t(i)} - (A_i - F_i)\frac{(F_i - \mu)}{N}\right)}{\sigma^4_{t(i)}} - \frac{(F_i - \mu)}{N^2\sigma^4_{t(i)}}\right).
\end{aligned} \tag{5.5}$$

If we take the limit of a single sample contribution, as in equation 5.6, we see that it converges to the gradient of the contribution of a single sample in the regular mean squared error, scaled by the variance of its stage.

$$\begin{aligned}
\lim_{N\to\infty} NT \cdot \frac{\partial MSSE}{\partial F_i} &= \frac{(A_i - F_i)\left(-2\sigma^2_{t(i)} - \cancelto{0}{(A_i - F_i)\frac{(F_i - \mu)}{N}}\right)}{\sigma^4_{t(i)}} - \cancelto{0}{\frac{(F_i - \mu)}{N^2\sigma^4_{t(i)}}} \\
&= \frac{2(F_i - A_i)}{\sigma^2_{t(i)}}.
\end{aligned} \tag{5.6}$$

In the figure 5.3 below we have the value of the loss function for the alignment example between a target distribution and a linear transformation (i.e., $\hat{x} = \sigma x + \mu$), for different sample sizes. We see that the function behavior is quite similar for all sample sizes.

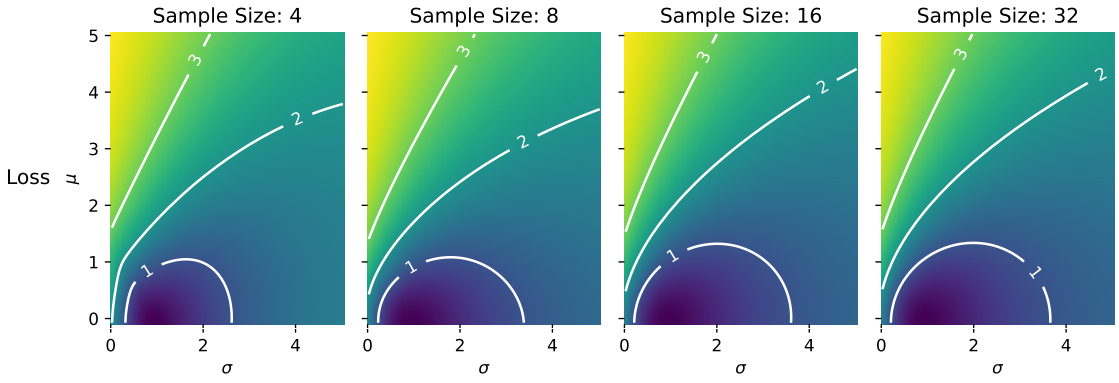


Figure 5.3: Loss function in the alignment example.

### 5.2.2 Domain Sampling

If we were to use only simulated states, as in SDDP, we would encounter two problems:

1. The initial weights of the network could be such that for all simulated states, independently of the scenario, its responses could be the same, slowing the training process;

2. By exploring only a small region of the state space, the policy might generalize poorly.

Both problems hinder a well behaved evolution of the network, possibly letting it to be stuck in a region with small gradient. To solve this problem, as seen in the algorithms 5, 6 and 7, we can use a sample of states over the domain to add exploration to the algorithm. A specialist can determine specific regions of the state space that must be sampled in order to direct the model to have a good resolution in these regions.

### 5.2.3 Soft and Hard Networks

As we saw in the sub-section 3.1.3, the $k$-convex manifolds can be fitted with a smooth procedure. The fitted and factual (the one to be optimized by the solver) functions are going to be slightly different, but the training process is accelerated. The same is of our interest with ReLU networks.

It has been found experimentally that by optimizing the ReLU target networks, some instances diverged and fled from the desired future cost functions. A mechanism that helped this process was the use of the GELU activation function [50] during training. So, as in the case of the $k$-convex, the trained network and the network to be optimized in the subproblem are different, but the training process is facilitated. Another soft activation function could have been chosen in the place of GELU, such as the SiLU [50] and the Mish [73].

In particular for the ReLU/GELU networks, but also relevant for the optimization of the $k$-convex functions, it is that as we increase their complexity (e.g., increasing the number of layers), the two networks are going to differentiate more and more. For this, a penalty is added, the loss function between the hard network prediction (ReLU) and

the soft network (GELU), multiplied by a smooth regularization value. The same loss function, the MSSE, was used.

### 5.2.4 Problem Scaling

When it comes to learning from neural networks, it is advisable that both inputs and outputs are low-magnitude values. For this, it is beneficial to change the scale of the problems so that both inputs (state) and outputs (costs) are small. Emprically, badly scaled problems lead to bad policies. Most of the problems can be easily rescaled with simple multiplications by constants.

# Chapter 6

# Explainability

In this chapter we describe the techniques used to study the two halves of the proposed algorithm, the network to the left (the importance of the observations) and the network to the right (the future cost function). First, we will discuss how SHAP can be an important tool to explain the decisions made by the algorithm, for the two networks.

## 6.1 SHAP

SHAP is a powerful tool for understanding what a model takes into consideration to make a decision. Next we will see two ways of how it can be incorporated, the first to estimate the relationship between the states, which can be used by any MSSO method with future cost functions; and the second focusing on the conditional and memory versions of our algorithm.

### 6.1.1 State Interdependence

Whether with traditional methods for MSSO problems, or with the proposed here, we can study how the values of the states affect the future cost. In particular, as these algorithms have been trained on a specific region of the state space, we will simulate them and study how the sampled states affect the future cost.

Having the states and the estimated future cost functions, it is easy to use SHAP to get the explanations of each state value. By the definition of SHAP seen in section 3.4, we can think that these explanations are the marginal costs of each entry of the state at that specific point. As we have seen from the beginning of this work, we often have problems in which decisions are rarely made independently, in which a dimension of the

state affects and is affected by other dimensions. But how to quantify this

A recent tool used to quantify the asymmetrical relationship between two variables is the Predictive Power Score (PPS) [110]. PPS is asymmetrical, agnostic to data, and detects linear and non-linear relationships. The idea is simply to use a non-linear regressor (decision trees, boosting algorithms etc.) and using only one variable as a input to try to predict another variable. Then we see how it performs compared to a naive regressor.

Here we will adopt a similar methodology. After simulating the policy and obtaining samples $\{\text{State}, \text{SHAP of Future Cost}\}$, the unidirectional PPS is calculated. In our case, the algorithm XGBoost [22] was used. The idea is to check how much the state itself is able to determine its marginal cost, i.e., how independent of the other states it is. If we have a high PPS in all state entries of a stage, a operator could make a decision based on each state independently, which is much easier to interpret and explain to other responsible agents. Otherwise, the operator who has to make the decision should be more cautious, now that he knows that the decision on the state is complex.

### 6.1.2 Observation Importance

The addition of conditional and memory variants enables a more detailed analysis of the algorithms. Now it is possible to verify, through SHAP, how an observation affects the future cost function, which allows operators and decision makers to quantify the impact of hypothetical scenarios. A rural producer can estimate how the price variation of a input affects their long-term profits and a dispatch operator can investigate how an out-of-time drought would increase the cost.

In addition to quantifying the impact of a specific observation, it is possible to calculate the SHAP value of the hidden state that reaches a stage. We are in practice quantifying the impact of the observation adjusted to its past history, since the observations are not temporarily independent.

## 6.2 Future Cost Curvature

In addition to understanding the marginal future cost of states and observations at specific points, we can study the curvature of the manifold defined by the future cost function. From this analysis arise two questions: does the neighborhood of this state

change suddenly (instability)?; is the neighborhood of this state different from others and if so do I have to worry (incertainty)? We will build the step-by-step methodology, but the schema can be seen in the figure 6.1. These methods are not the exclusive to the proposed algorithms and can be used to study traditional algorithms of MSSO problems.

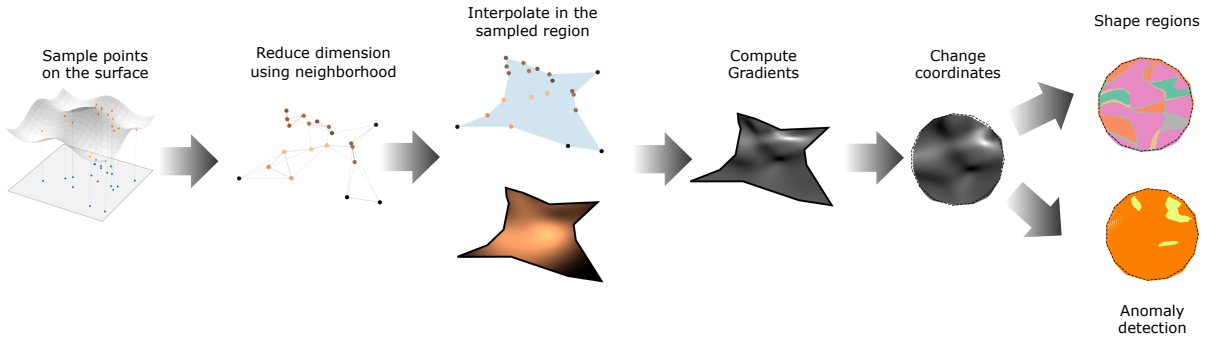


Figure 6.1: Curvature Study Schema. We sample points from the manifold in a uniform way over the domain. With manifold learning to construct a two dimensional representation. We interpolate the sampled region and compute gradients. A coordinate transform is applied for better visualization. Local shape and anomalous points are computed.

### 6.2.1 Processing

The first step is to sample points on the manifold of the future cost function to be studied. As the curvature depends on the second derivatives, calculating the curvature directly on this manifold, which has high dimensionality, is a computationally expensive process. The strategy adopted is to study the curvature on a surface, with reduced dimensionality. In higher dimensions we have incomprehensible geometric behaviors that when projected on a surface we certainly have loss of information. We expect that locally these multidimensional artefacts are similar to those of the surface.

So the second step is to use some manifold learning algorithm to get a representation of this manifold now in only two dimensions. We use LLE, as discussed earlier. The function image is kept for the subsequent steps.

With the two-dimensional coordinates, it is sought to find the region of this new space that corresponds to the likely region of the projection of other points from the original manifold. For this we use the $\alpha$-shape algorithm. With the image values of the future cost function, we can interpolate a grid over the entire region defined by $\alpha$-shape. The interpolation used was the radius-based function interpolation, with multi-quadratic function.

In this grid, we can estimate the first and second derivatives of each interpolated point and calculate the curvature attributes described in the sub-section 3.3. Here we already have the necessary data to try to answer the two questions. Only to facilitate viewing, we can transform this study region into a regular polygon using mean value coordinates (sub-section 3.3.6).

### 6.2.2 Local Shape

As we saw in the table 3.1, we can characterize the local behavior of a surface through its two principal curvatures. So we know if locally the surface is approximately a linear, concave, convex, hyperboloid etc. With real problems, it is not always possible to run the optimal solution obtained and it is interesting to know if disturbances in its neighborhood have sudden changes on the value of the future cost. If we are in a hyperbolic region of the surface and we don't have sufficient precision to run exactly the optimal solution, it can be better to choose a sub-optimal solution in a more linear region. So we get a map of surface stability of the future cost function.

### 6.2.3 Anomalous Region

The surfaces of the future cost functions learned may have different resolutions throughout the state space. Occasionally, a scenario can lead the optimal solution chosen to low-resolution regions not because it represents the real optimal solution but rather because it is a little explored region that, during training, the model erroneously extrapolated. Thus, we would like to find regions of the surface of the future cost function that present characteristics that differentiate them from the others, in the hope of finding these poorly conditioned regions.

The strategy adopted is to consider each point of the interpolation as a sample, and each curvature attribute as a feature. Then we run the anomaly detection algorithm Isolation Forest [66]. It selects a feature by chance and randomly chooses a value in the interval between the minimum and maximum as a decision node of a decision tree. The number required to isolate a sample, aggregated by the average of a forest of these trees, is an indication of how different this sample is from the rest. When a sample consistently needs few decisions to be isolated, it is an indication of whether it is an anomaly.

# Chapter 7

# Experiments and Results

In this chapter we put the algorithm to the test in order to verify three things: whether it converges to the correct functions in simple problems; whether it proposes robust solutions in real systems; and how its hyperparameters affect its performance.

## 7.1 Control Problem in One Dimension

The definition and modeling of the Control Problem in One Dimension is given in the section 4.1. Because it is a pretty simple problem, with the state and uncertainty variables both have only one dimension, it is possible to calculate the expected future cost functions through the recurrent definition.

Let $m \in S(n)$ be a scenario from the set of scenarios at stage $n$. We have that

$$\bar{Q}_n(x_n) = \begin{cases} 0 & n = T \\ \sum\limits_{\xi_m \in \mathcal{S}(n)} C(x_n + \xi_m) + \beta \cdot \bar{Q}_{n+1}(x_n + \xi_m) & n < T \end{cases} \tag{7.1}$$

As can be seen in figure 7.1, the future cost functions are non-convex in all stages. This is because the discontinuity $c_t$, in which the immediate cost $C$ id given by $\min\{|u - 1|, |u + 1|\}$, in which $u = x_{t-1} + \xi_t$.

Note that since each stage has its own sample of uncertainty, even if they com from the same distribution, the estimated expected future cost curves are not exactly the same but for a constant, they have certain irregularities. As the subsequent experiments also use the same sample, it will be, for our use, the real expected future cost function. As we increase the sample size, these functions converge to the problem optimum.

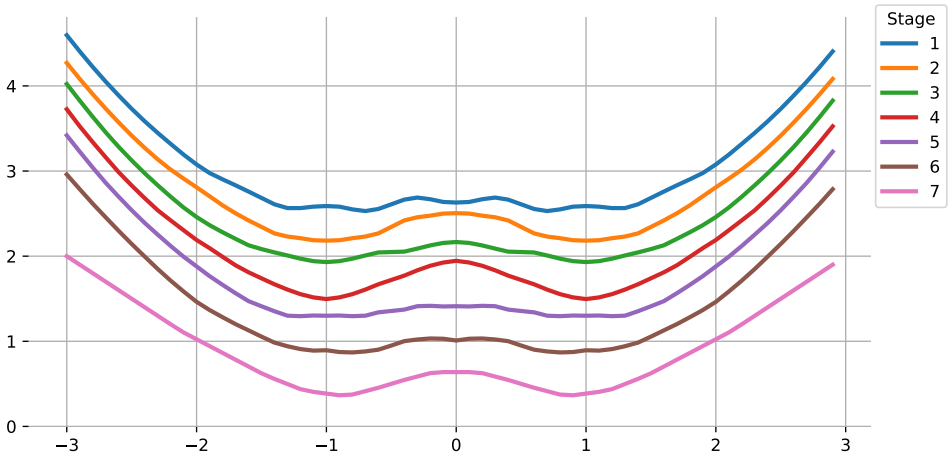


Figure 7.1: Optimal Expected Future Cost Functions $\bar{Q}_n$ for the Control Problem in One Dimension.

### 7.1.1 Convergence to the Mean

The first experiment is to verify whether the algorithm is able to estimate the expected future cost function. As it is a problem without temporal dependence, we will limit ourselves to the simplest version of the algorithm, the convergence to the mean. The experiment was done with a network to right (the future cost function) with two hidden layers with 40 neurons each; with 20 random samples in the interval (-3,3); 10 samples of forward pass, using the same sample of the figure 7.1; smooth regularization of 0.2; by 100 iterations. The figures 7.2 and 7.3 illustrate, respectively, the hard and soft future cost functions learned.

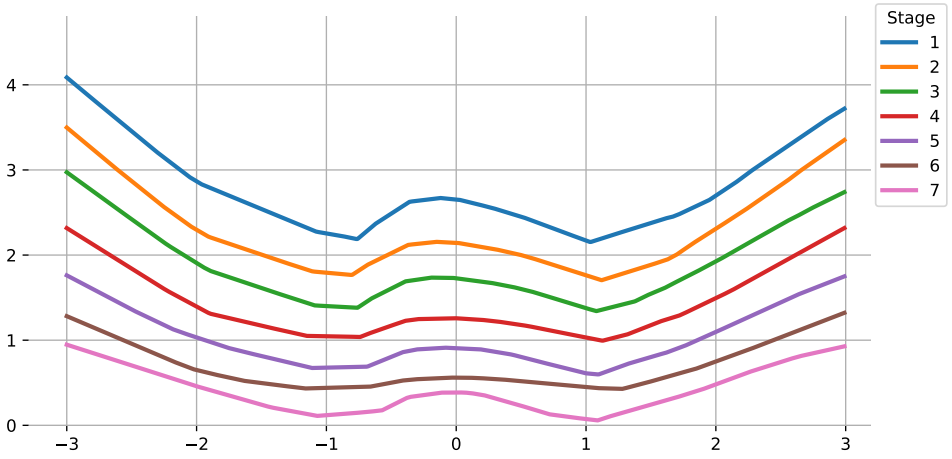


Figure 7.2: Hard Expected Future Cost Functions $\bar{Q}_n$ for the Control Problem in One Dimension.

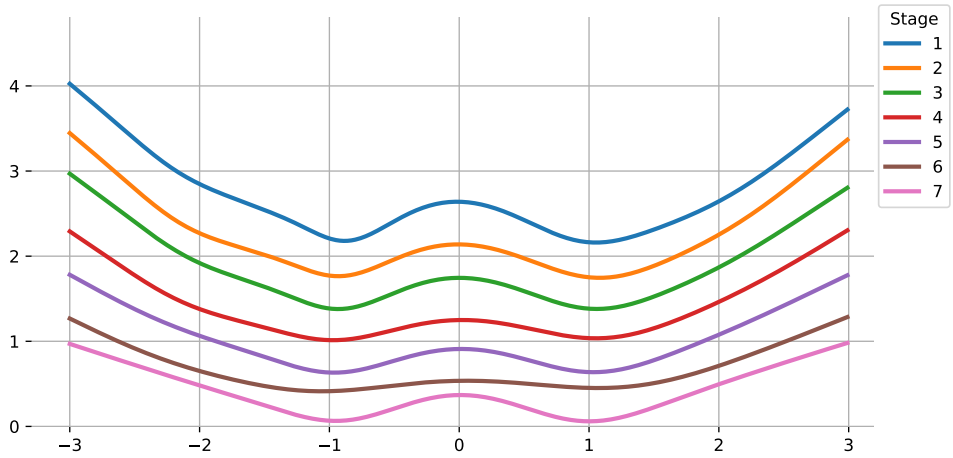


Figure 7.3: Soft Expected Future Cost Functions $\bar{Q}_n$ for the Control Problem in One Dimension.

The figure 7.4 illustrates the expected future cost functions in different iterations of the same training, thus illustrating the evolution. Note that in the 15th iteration the algorithm is still learning the magnitude of each function; and that in the 30th iteration the functions already possess the approximate shape of the actual functions. From the 50th iteration onwards there is little visual improvement.

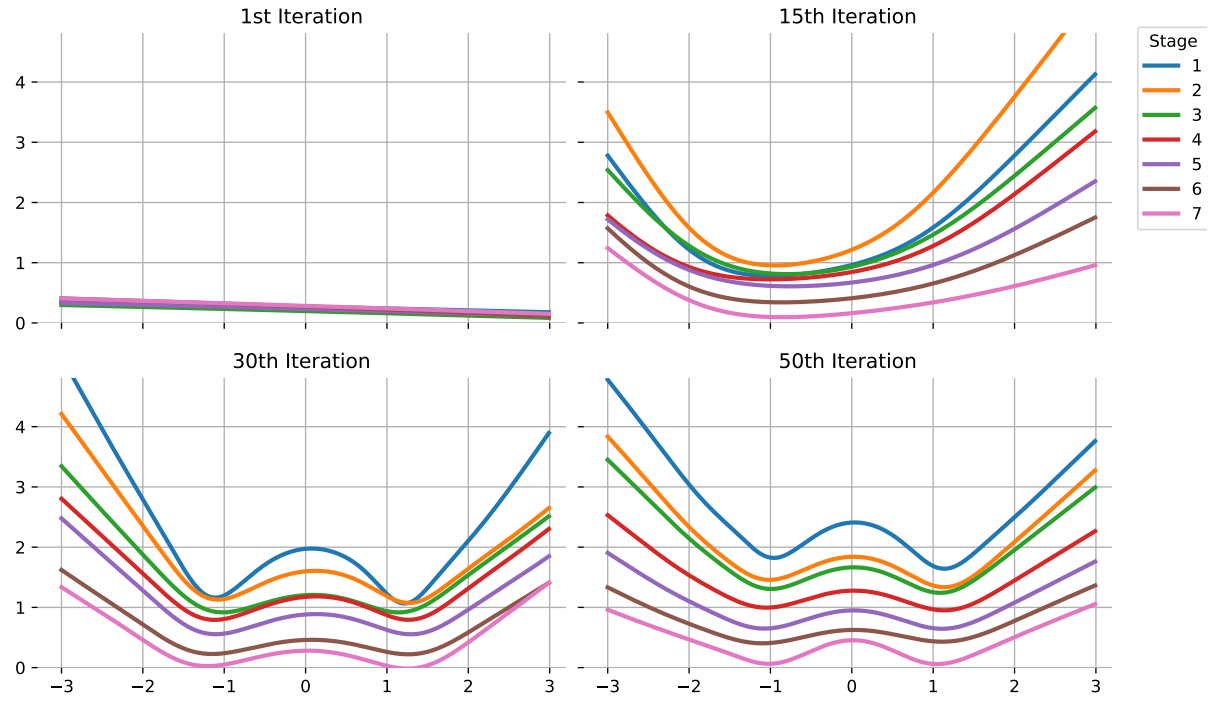


Figure 7.4: Soft Expected Future Cost Functions $\bar{Q}_n$ evolution for the Control Problem in One Dimension.

Even if there is no temporal dependence between the stages of the problem, the use of the conditional algorithm can still make sense. In the figure 7.5 we have the hard and

soft conditional future cost functions conditioned in the greatest and the lowest value of the scenarios, i.e., the two most distinct observations possible. We see that the functions are little affected: we can use the conditional algorithm, which as we saw in the subsection 5.1.2 is $S$ times faster, to learn policies that have no temporal dependence.

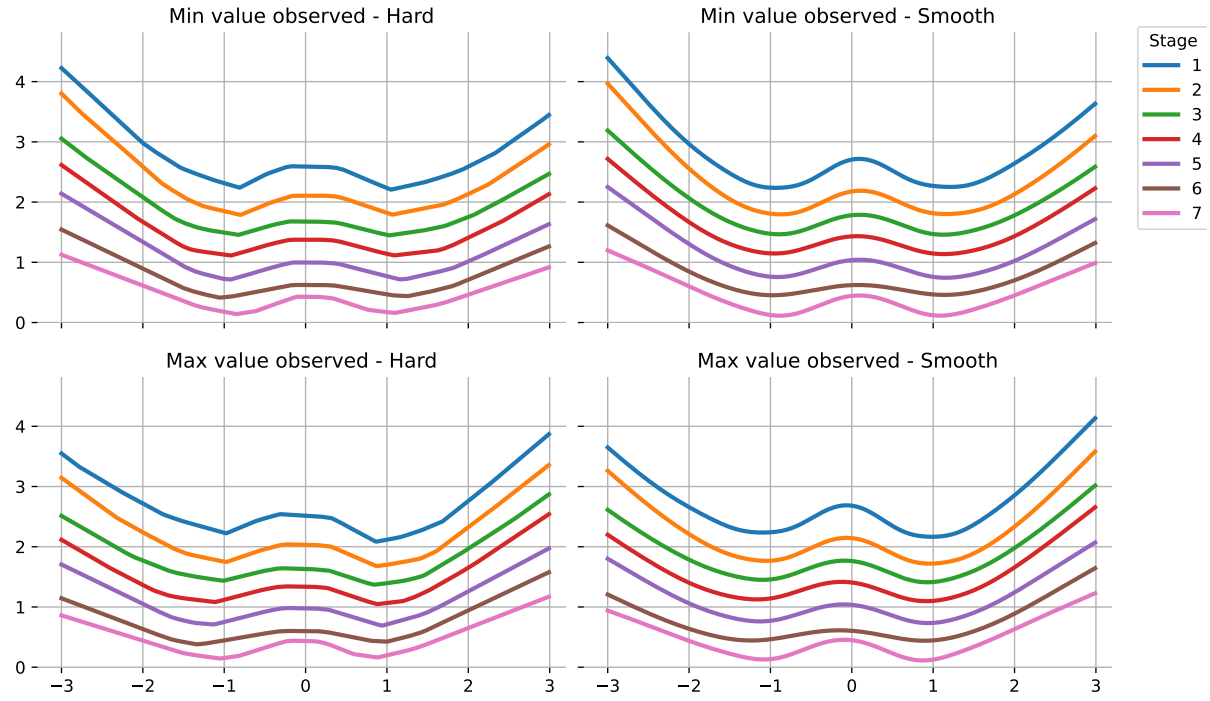


Figure 7.5: Conditional Future Cost Functions $Q_n|\xi_n$ for the Control Problem in One Dimension.

### 7.1.2 Convergence for Different Networks to the Right

One important decision to take during the training process is how to choose the network architecture to the right. As we saw, the network can be a $k$-convex manifold or a ReLU network. We have the loss function for different architectures in figure 7.6. In it, the experiments were carried out with 10 forward pass samples (policy simulation) and 20 random over the domain. It was carried out 3 times and taken the average. In the first graph, which shows the average of the loss function of all the $k$-convex architectures and all the ReLU networks (non-convex), we have the behavior of the loss function between the classes of network is quite different. $k$-convex networks have a smaller initial loss, but decay more slowly than those of the non-convex networks.

When we see the second chart, in which the average is decomposed to the different

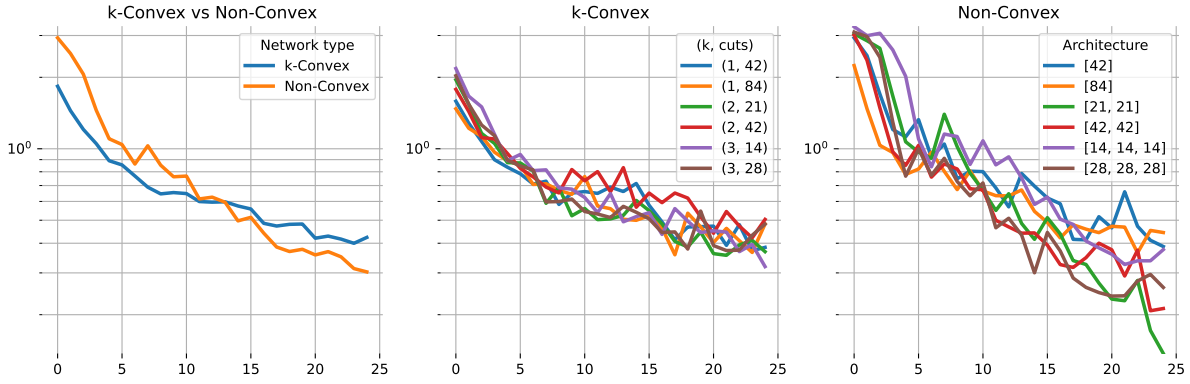


Figure 7.6: Loss functions throughout 25 iterations for different $k$-convex and non-convex architectures. Each architecture was trained thrice, and the average is shown above.

combinations of $k$ and number of cuts, we see that the loss function has approximately the same behavior, with little variation between the models. Because it is a problem with non-convex future cost functions, we would expect that the loss function with $k = 1$ would have a worse behavior, but it is not what was observed. Its functions are shown in figure 7.7 below. See that 25 iterations were not enough for the model to learn the functions correctly, especially for stage 1. Besides, it still represents well the optimal functions.

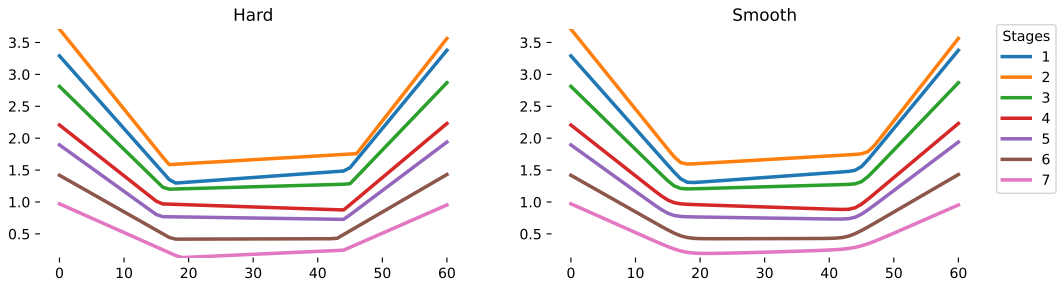


Figure 7.7: Hard and Soft Expected Future Cost Functions with $k = 1$ $\bar{Q}_n$ for the Control Problem in One Dimension with $k = 1$ and 42 cuts after 25 iterations.

The third chart already shows something different. The total number of neurons is not a good predictor for the performance of the network, since there are situations in which networks with the same number of layers but a smaller number of neurons per layers win and situations that they lose to the largest. Distinctively it is noted that networks with two hidden layers have a better loss function than the others, with the networks with three layers coming in second place. If on the one hand the increase of layers improves exponentially the spatial resolution of a ReLU function, it also more easily increases the difference to its soft representation (GELU), which can lead to a breakdown between the network learned and the network fitted. This would be an explanation for the lower performance of the network with three layers in the experiment. A solution could be to increas the value of the smooth regularization. Graphically, all functions are

approximately linear, which, considering the log scale, means they have approximately exponential behavior.

To see how the functions generalize (thanks to a large number of samples on the domain), the squared difference between the future cost functions learned and those of the figure 7.1 was studied. The results, for some selected stages, are in figure 7.8. In the first column we have the results grouped by network class, while in the rest are the specific architectures of each class.

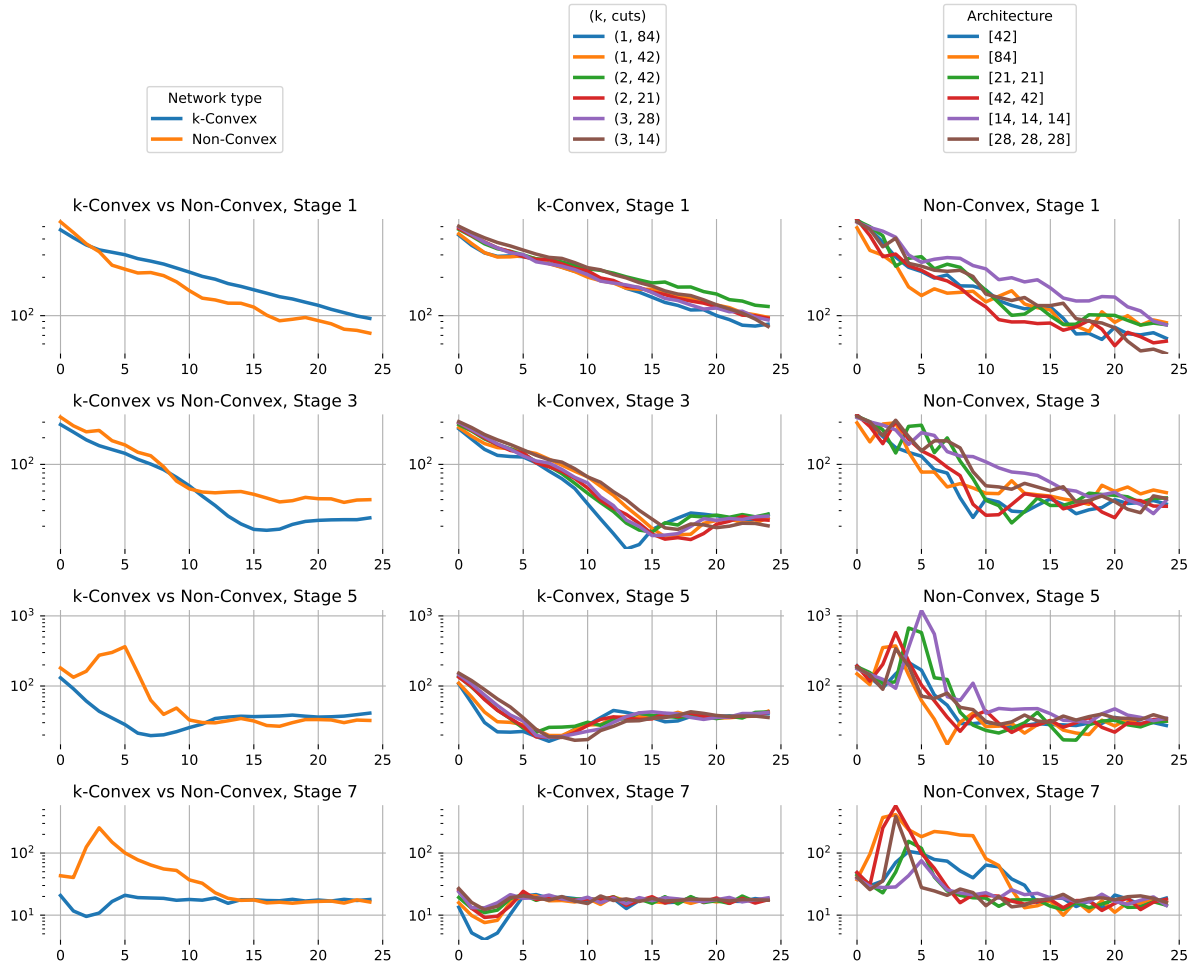


Figure 7.8: Squared differences between the estimated future cost functions and those of figure 7.1 for 25 iterations for different $k$-convex and non-convex architectures. Each architecture was trained thrice, and the average is shown above.

On the first column, we see that the convergence behaviors between the classes and between the stages are quite different. On stage 7, which has lower magnitude values, random cuts already have a good approximation to the future cost function and the model quickly converges to a function close to the real. Interestingly the squared difference initially falls below the value that eventually the function converges to, showing a slight over-fitting to the training region. Non-convex networks take longer to converge at this

stage, significantly increasing in the initial iterations. This can indicate that the gradients of the other stages, even with the reinforcement of the algorithm for the latter stages, is greater. Eventually the squared difference begins to converge to a value close to the $k$-convex one. Stage 5 has a similar behavior to the 7th one, but with a slower convergence on the $k$-convex networks.

Stages 1 and 3 are different. Here the non-convex networks have no increasing phase seen in the subsequent stages. In stage 3, the $k$-convex networks converge even later, with the same behavior of converging above the minimum. The non-convex networks converge to a different value, higher. Stage 1 is the only stage in the graph in which the non-convex networks lead with some margin almost from the beginning. None of the networks classes seem to have converged, more iterations would be needed, as it has already been advanced in the figure 7.7 in which we see that stage 1 is not still well positioned.

As for the other two columns, the behavior is similar to the average of the classes, again the non-convex networks have more variance. On the $k$-convex networks, the ones with $k = 1$ seem to converge faster, mainly in the final stages. Considering that they have the least numbers of parameters, it makes sense. As for the non-convex networks, in stage 7 the shallow ones have difficulty learning, while in the final stages the networks with 3 layers are the ones that have difficulty, especially that of 14 neurons in each. Those with two layers do well in all stages.

Finally, figure 7.9 has the mean execution times for each iteration for each network to the right architecture. We see that both for $k$-convex and non-convex networks what affects the time the most is the total number of parameters (cuts and neurons) and not the complexity ($k$/layer number). The latter does affect the execution times, but in a less intense way. As expected for what was discussed in the table 5.1, non-convex networks have a greater complexity and therefore require more time to be optimized. Note that in the experiment were used 30 samples (10 of simulation and 20 over the domain) and therefore the cost to solve a single instance is much lower.

### 7.1.3 Hyperparameters and Convergence

We can also study how it is the learning process with different combinations of hyperparameters: the number of samples in the forward pass, the number of random samples over the domain and the smooth regularization parameter. We have the loss function for each each combination in the figure 7.10. Note that in situations with 10 random samples over the domain (i.e., the random signal is greater or equal to the signal of the policy simulation) there is convergence of the models, with loss function values

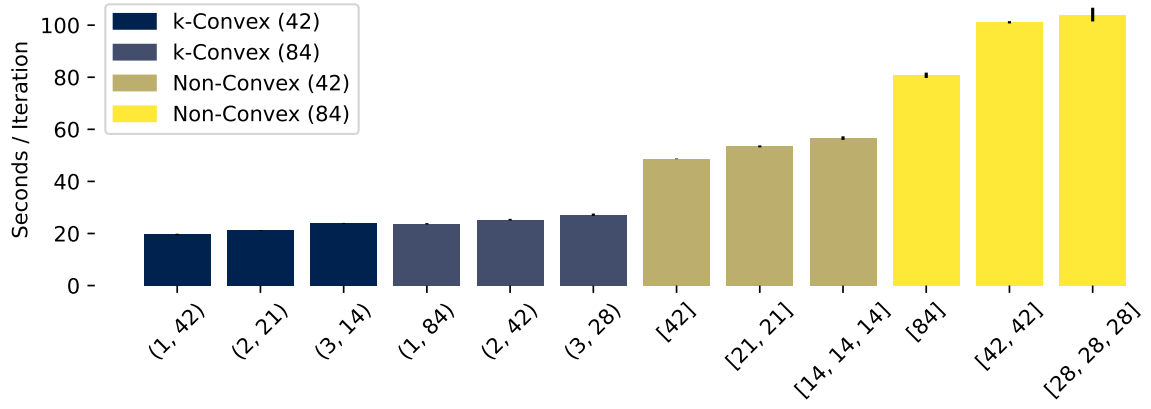


Figure 7.9: Execution times in the architecture experiment, per iteration, for different $k$-convex and non-convex. Each architecture was trained thrice, and the average is shown above.

close to zero. The model with only the simulation signal (forward pass) in purple does not always converge (with this many iterations), but they have a downward trend. Compared to others, it initially has a slower drop. The red model has just a few random samples on the domain (5) compared to the number of simulated samples (10), having an accelerated drop but a high and stable final loss. A positive value for the smooth regularization seems to accelerate the initial fall of all models.

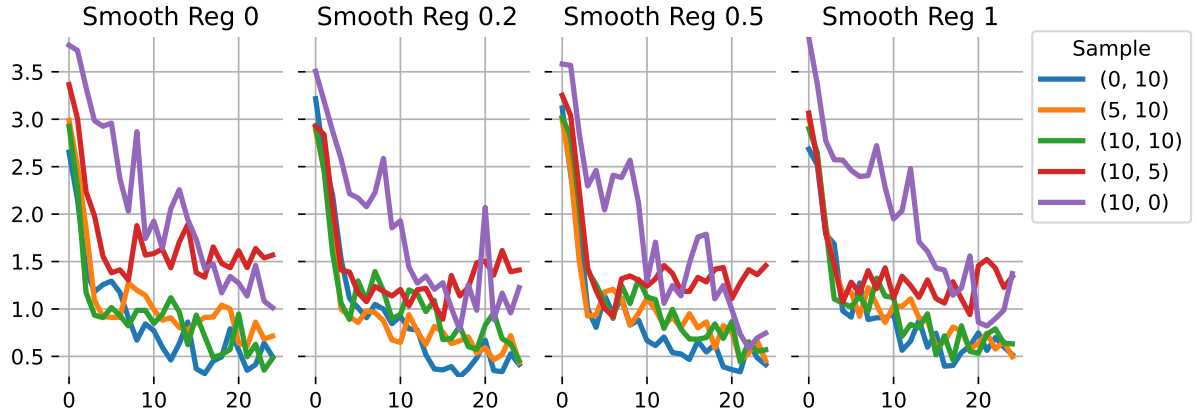


Figure 7.10: Loss functions for 25 iterations for different combinations of number of forward pass samples, number of random samples over the domain and smooth regularization factor. The network to the right has two hidden layers with 40 neurons each. Each combination was trained thrice, and the average is shown above.

Being a parameter that increases the exploration region of the state space, it is interesting to investigate how generalization is affected by the number of random samples over the domain. For this, the squared difference of the estimated expected future cost functions and those of the figure 7.1 was calculated. As the simulation begins with an

initial state of value 2, it is very rare that we get to explore a negative state situation. We have the results in the figure 7.11, for some chosen stages.

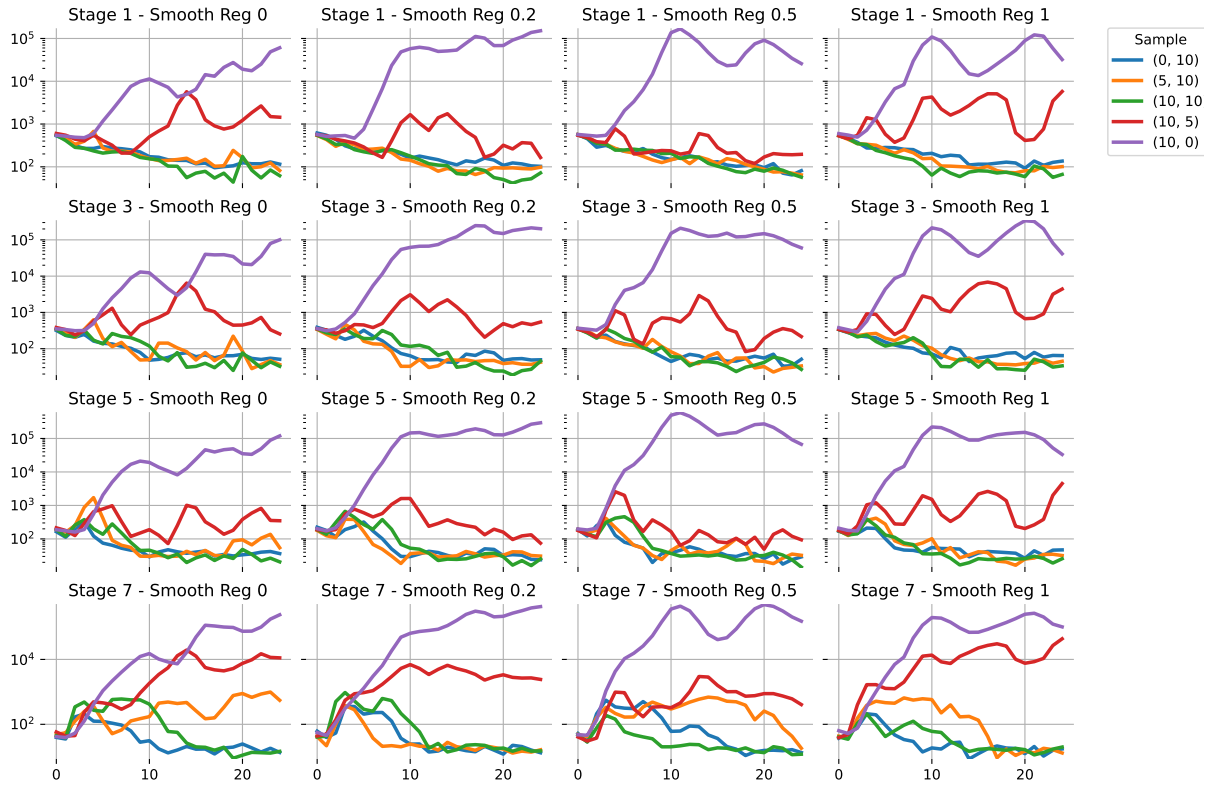


Figure 7.11: Squared differences between the estimated functions and those of figure 7.1 for 25 iterations for different combinations of number of forward pass samples, number of random samples over the domain and smooth regularization factor. The network to the right has two hidden layers with 40 neurons each. Each combination was trained thrice, and the average is shown above.

In purple we have the parametrization with only simulated signal (forward pass). We see that the squared difference grows with the iterations instead of decreasing. The model suffers from over-fitting in the explored region, of positive state values. In red we have a situation with a little signal from random samples. The squared difference seems to grow and stabilize, but in a level quite below the purple one. All the others, with the exception of stage 7 with regularization equal to zero, seem to converge to a low squared difference: good generalization.

Note that in some stages, such as stage 7, the squared difference does not (ignoring oscillations and considering just the trend) monotonically decrease. If first increases to then drop.

With the exception of the purple parameterization, which explodes quickly, the soft regularization with values of 0.2 and 0.5 seem to offer a better behavior (more stable, lower magnitude) than the zero value and the value of 1.

## 7.2 Hidrothermal Dispatch Problem

The definition and modeling of the Hydrothermal Dispatch Problem are given in section 4.2. Unlike the Control Problem with One Dimension, the Dispatch Problem is complex enough so that the recursive approach to estimate the expected future cost function becomes intractable. Therefore, the experiments of this section will not be compared with a optimum, but with the model that guarantees convergence for convex problems: SDDP. It was trained with 100 cuts.

The work has numerical results for a real hydrothermal system, the Colombian system, consisting of 72 hydroelectric and 32 thermal plants. The system has 80 historical scenarios.

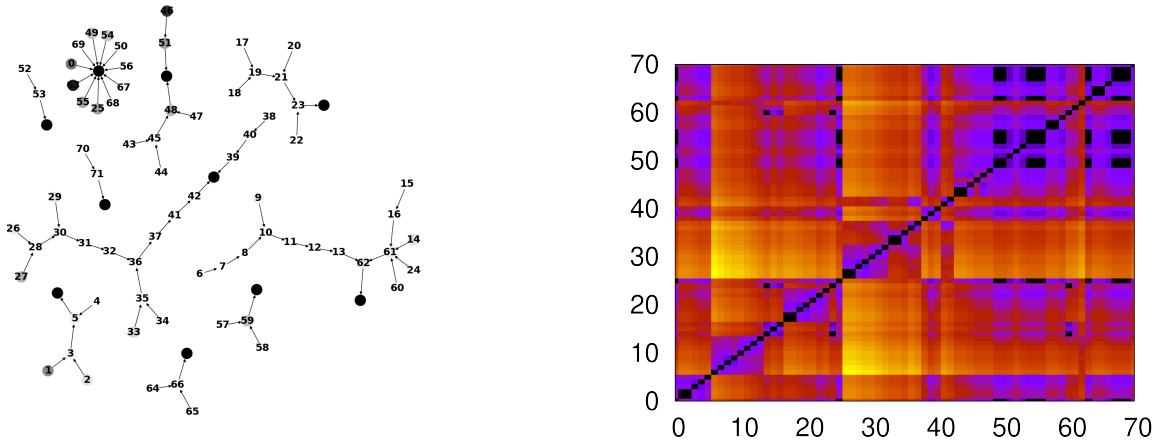


Figure 7.12: **Left:** The reservoirs are arranged following the topology of the river basins. The color of a vertex corresponds to its total volume capacity. Vertices in black are a river mouth. **Right:** Distance Matrix between each pair of reservoirs. Darker points are topologically similar pairs.

Each hydroelectric plant has a reservoir. These reservoirs are organized according to the topology in Figure 7.12. The figure also shows the matrix of distances between each pair of reservoirs. The expected monthly demand, historical scenarios of rain inflow, and all the constraints of reservoirs were made available by PSR (https://www.psr-inc.com/), and are illustrated in the figure 7.13 below. The operation period is 2 years divided into 24 decision stages.

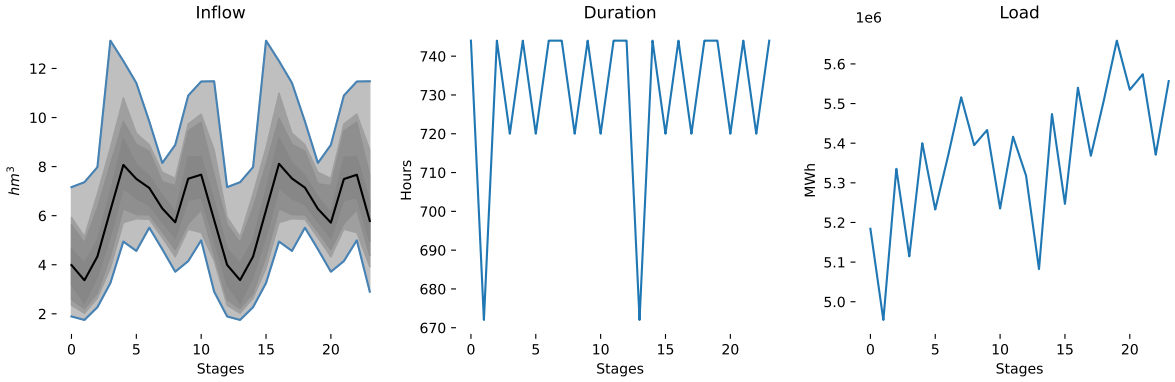


Figure 7.13: Dispatch Scenarios. First graph shows the inflow distribution, the second shows the length in hours of each stage and the third shows the demand over time. The last two are deterministic.

### 7.2.1 Algorithmic Details

Here we have experiments aimed at studying how different algorithm modifications affect convergence and training. We will study the conditional and memory versions of the algorithm; whether the pass through the data is in batches or by samples; and what is the ideal frequency of updating the network weights, at each stage or only at the first (which is the last in the backward pass). Figure 7.14 below brings the results. `All` and `Last` refers to the frequency of updating; and `Batched` and `Sampled` refers to how the policy simulation data are selected, in order or randomly.

We see that the conditional version and the memory one have different behaviors. For the algorithm version in which we update the network at each stage, in green and blue in the graphics, the two strategies quickly converge to a local minimum, regardless of the type of sampling method. As for the frequency of update, we see a difference. While the conditional model can converge to a lower level, the use of only one update by iteration makes the model with memory diverge. The use of random sampling of scenarios takes the conditional model with only one update to a level even lower than the use of the data in ordered batches.

In addition, we see that in the conditional model the loss function and the total cost are highly related when we have a single update.

In the figure 7.15 above we have the average execution time of the algorithms. The use of memory and the use of ordered batches increase the time.

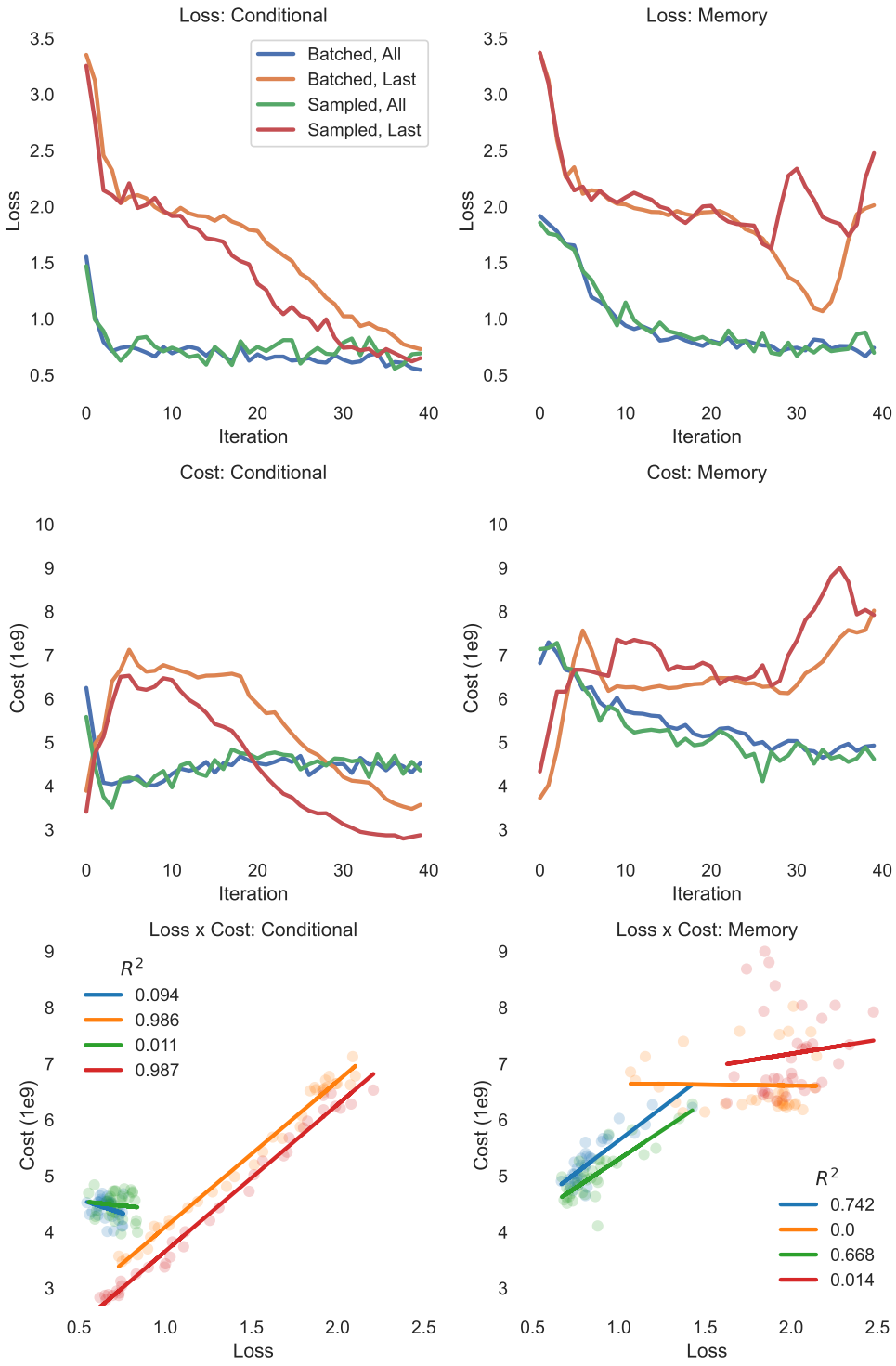


Figure 7.14: On the left we have results with regard to the conditional version, and on the right with regard to the version with memory. The first row illustrates the loss function along the iterations. The second illustrates the total cost of the policy along the iterations. The third shows the relationship between the loss function and the total cost. All experiments have been repeated 3 times and what is shown is the average. Parameters: soft regularization of 0.05; a single segment of 100 cuts; batch of size 20; 10 random samples on the domain.

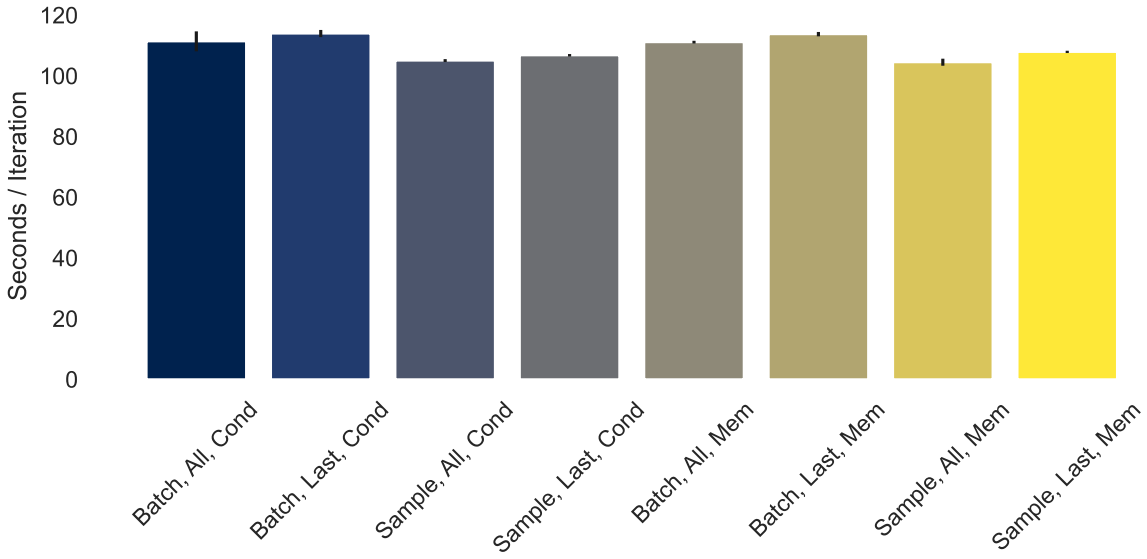


Figure 7.15: Execution times for different versions of the algorithm. `Batch`: the scenarios are simulated in ordered batches; `Sample`: the scenarios are sampled randomly; `All`: the network is updated at each stage; `Last`: the network is updated only in the last step of the backward pass; `Cond`: conditional model; `Mem`: model with memory.

### 7.2.2 Policy Performance

We will now discuss how the algorithm performs on this dispatch problem. As it is a convex problem, our baseline is the performance for SDDP, which was $1.27 \times 10^9$\$. As seen in figure 7.14, for a network to the right using 100 cuts (convex), the best policy was that of the conditional algorithm with random samples and single updates. In the figure, it is not clear if the algorithm has already converged and therefore we can train it for more iterations. As we see in the figure 7.16, when the algorithm converges, it still has a much higher expected total cost than that of SDDP, even using the same number of cuts. So, SDDP learns a better use of these cuts than SNDP.

As though it is a problem with convex future cost function, we can see how a non-convex policy performs, with the use of a ReLU network to the right. In figure 7.17 we have the evolution for the conditional algorithm with a ReLU network to the right with architecture with layers $40, 40$. We see that the expected total cost comes very close to the obtained by SDDP. Thus, we can investigate other architectures for the network to the right and see how they approach the cost of SDDP. A observation regarding the figure 7.17 is that it is not clear by the loss function whether the policy has already converged or not.

This research regarding the performance of different architectures is illustrated in figure 7.18. Unlike the previous two, here we show the loss and the minimum cost so

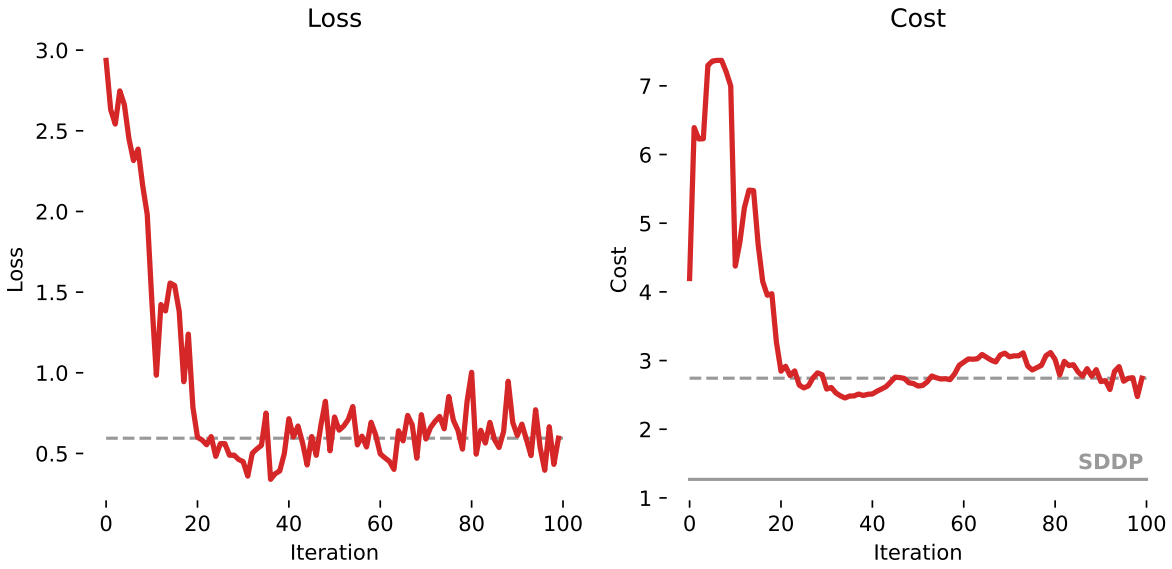


Figure 7.16: On the left we have the loss function over the iterations. On the right we have the total cost of the policy over the iterations. Parameters: smooth regularization of 0.05; a single segment of 100 cuts; 20 forward pass samples; 10 random samples on the domain.

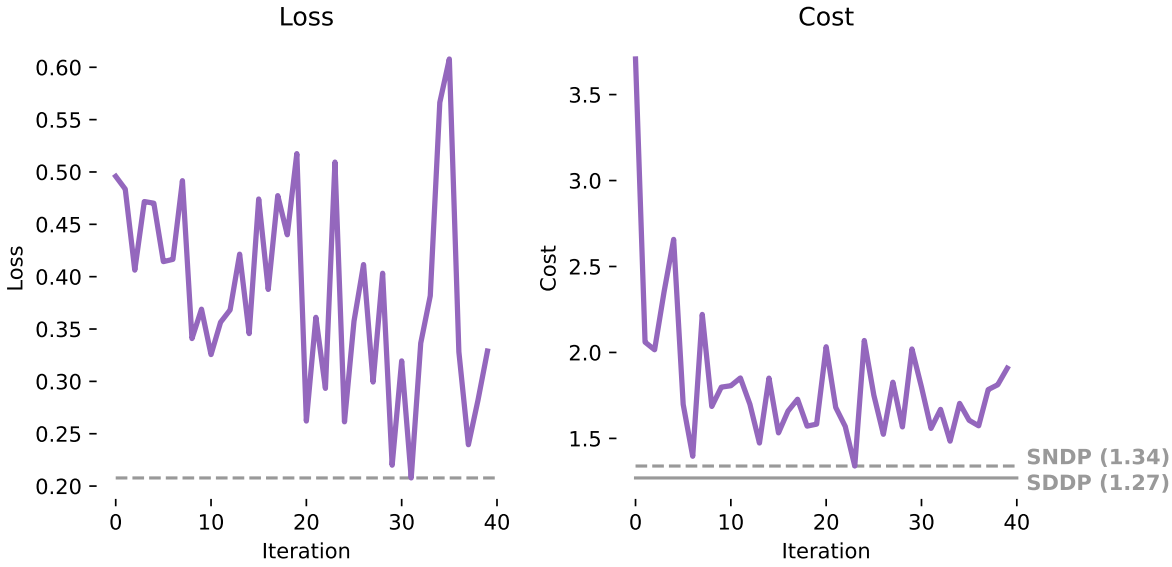


Figure 7.17: On the left we have the loss function over the iterations. On the right we have the total cost of the policy over the iterations. Parameters: smooth regularization of 0.05; ReLU network $[40, 40]$; 20 forward pass samples; 10 random samples on the domain.

far observed throughout the training. This choice was given by the stochastic nature of the training. In the first figure we see that policies converge to close values in the loss function, with the conditional policy $[20, 20]$ apparently have full converged. In the second image we see the same behavior, with the architecture with memory closest to the value

obtained by SDDP. Unlike SDDP, we see in the last figure that neural policies prefer a high initial cost and a lower cost in the second peak.

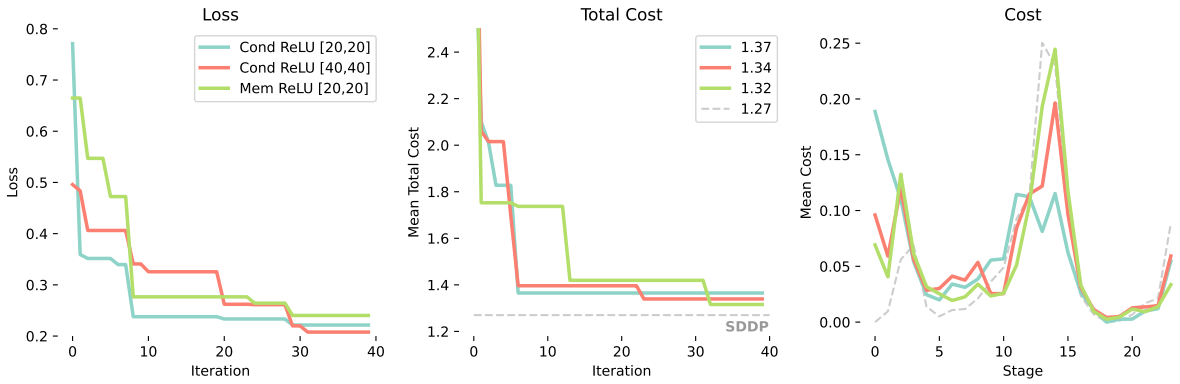


Figure 7.18: On the left we have the accumulated minimum loss function along the iterations. On the center we have the accumulated minimum total cost. On the right we have the average cost per stage. On grey we have the SDDP policy. Parameters: smooth regularization of 0.05; conditional ReLU network $[20, 20]$, $[40, 40]$ and with memory $[20, 20]$; forward sample size 20; 10 random samples on the domain; update at every stage.

We can now investigate these policy differences. In figure 7.19 we have the distribution for many different variables over the simulation of all scenarios using the different policies, including the optimal policy (where there is no uncertainty) and the greedy policy (the immediate lowest cost). We studied total volume, turbined volume, spilled volume, thermal energy generation and cost.

Let's go by steps. On the first row we have the distribution of total volume. Although they differ for specific reservoirs, SDDP has a total volume distribution very similar to the optimum one. The neural policies follow the pattern somewhat, specially the network with memory. For the turbined outflow volume, all trained policies stay around the optimum region, but again SDDP is the closest to the optimum. Spillage is where the neural policies do not deliver, with some scenarios causing the waste of high volumes of water, specially at the ending stages.

On thermal generation we see that the optimum policy has quite a different behavior compared to the others, with no peak. All learned policies peak around the same time, but the neural ones differ from SDDP. The latter has a brief moment around stage 5 with almost no thermal generation, which does not happen on the neural policies. The total cost is similar to the thermal graph, but we see that SDDP is worse than the neural policies for a specif scenario, showing a huge peak in costs. Considereing that the mean total cost of all learned policies do not differ that much, is interesting to see how these learned policies are so unique.

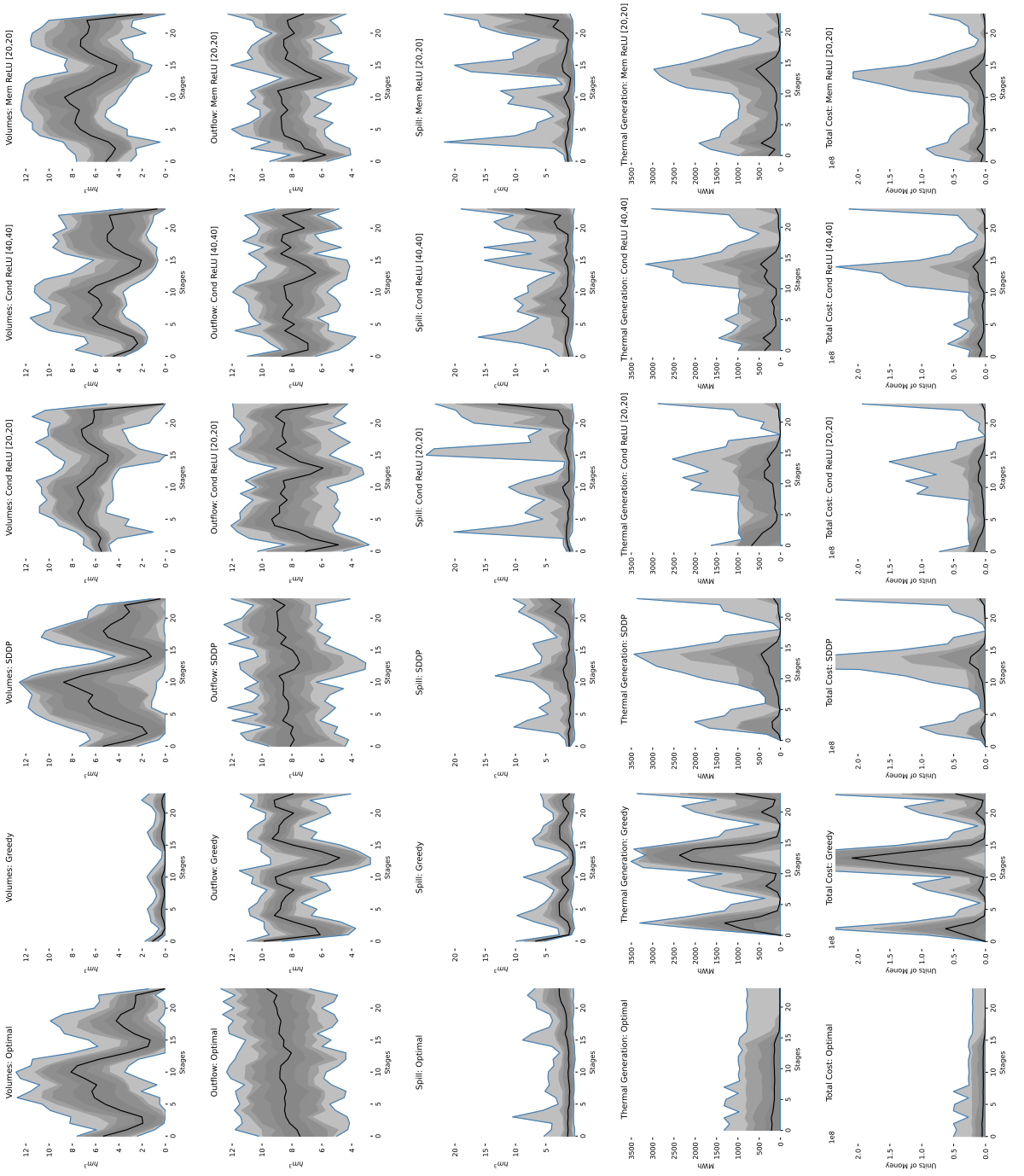


Figure 7.19: Summary graphs of the simulations for the dispatch problem for the policies optimal, greedy, SDDP with 100 cuts, conditional SNDP ReLU $[20, 20]$, conditional SNDP ReLU $[40, 40]$ and SNDP ReLU with memory $[20, 20]$. Grey regions represent the intervals between the quantiles of $(25\%, 75\%)$, $(10\%, 90\%)$, $(5\%, 95\%)$ and $(\min, \max)$.

### 7.2.3 Robustness on Extreme Scenarios

In addition to the average performance, we can study how the algorithms perform on gradually more extreme cases. In figure 7.20 below, we have the policies learned dealing

with new scenarios. The scenarios represent the average of the $\beta$ fraction of the scenarios with the lowest total influx volume. If $\beta = 1$ then we have the average of all scenarios and if $\beta = 0.5$ then our new scenario is the average of the half of the scenarios with the least water.

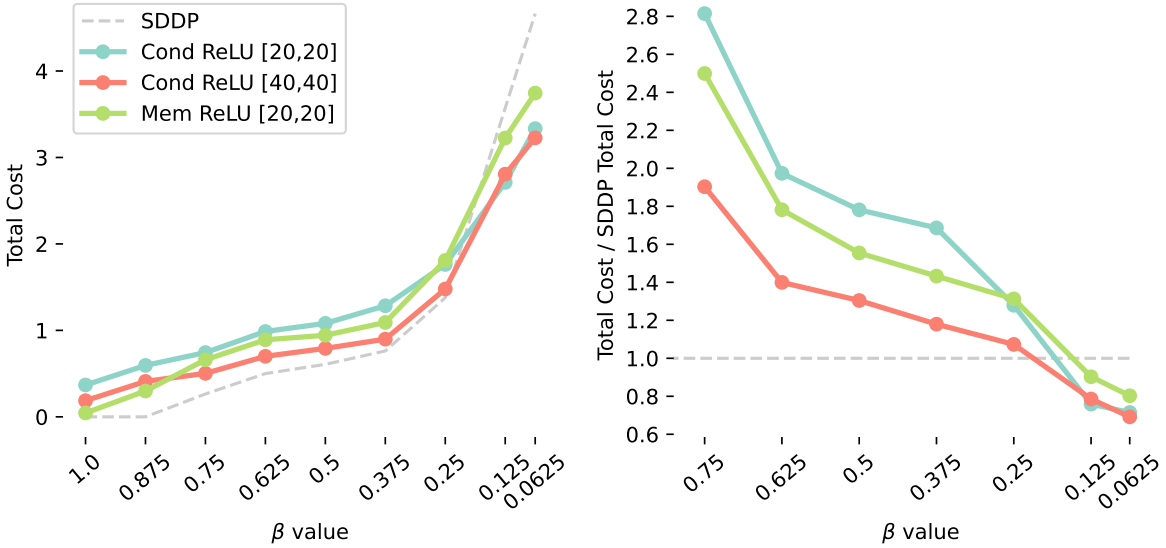


Figure 7.20: On the left we have the cost of the policies. On the right we have the ratio between the cost of the policy and the cost of SDDP. As SDDP has zero cost for high values of $\beta$, the x axis begins with 0.75. The scenarios represent the average of the $\beta$ fraction of the scenarios with the lowest total inflow volume.

We see that SDDP learns a zero-cost policy for the scenarios similar to the average, slowly increasing its cost until the final quarter, where the cost grows rapidly. Thus, SDDP learns a good policy for the average, not for the tail, what is expected since it was its risk measurement. For the SNDP policies the opposite occurs: higher cost compared to SDDP for average scenarios, and lower than SDDP for the tail. One of the policies comes to be almost 70% of the cost of SDDP. Interesting to observe that for $\beta \leq 0.75$ the policy with memory is dominated by the conditional $[40, 40]$.

### 7.2.4 Model Explanations

In this subsection we will use the techniques described in section 6.1 to analyze the policies. As described there, we can calculate the importance of states, observations and memory vector for the different policies. In figure 7.21 we have the absolute importance aggregated by each of these categories.

The importance of the states follows the same "corcovado" shape in all policies, with a gradual rise all the way up to a peak, around half of the planning period, and a sharper fall. The magnitudes are similar, with the conditional model $[20, 20]$ having a peak higher than the others.

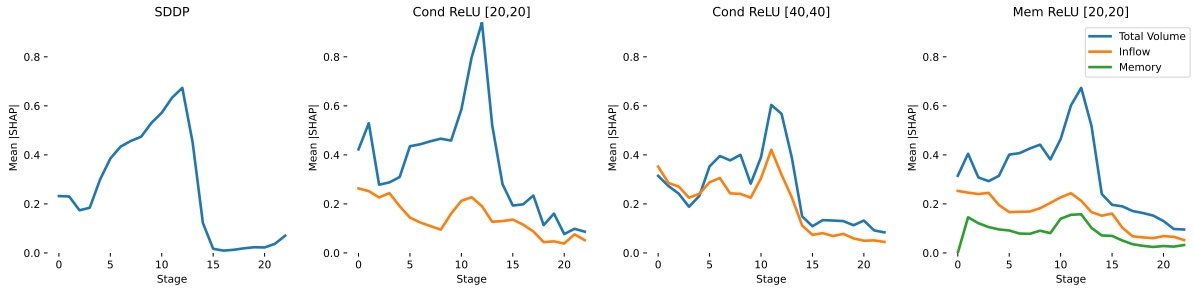


Figure 7.21: Input importance for the future cost function of different policies.

This information is relevant for lawmakers and managers, in which a specific reservoir may have a greater (in absolute value) importance than the others, requiring special attention. In real situations it is not always possible to follow the policy and therefore it is useful to keep in mind those reservoirs of greater impact on future costs.

As for the importance of the inflow, we see that the three policies actually use this information to adjust the future cost. The conditional policy $[40, 40]$ even gives more importance to the inflow observed at the beginning of the period than the current volume of the reservoirs. If the different scenarios already differ from the beginning, the policy can learn to identify the corresponding future and prepare for it at the present. Beyond the peak, the importance of the inflow for the three policies decreases. Following the previous hypothesis, as we have less future to worry, the importance of identifying in which type of scenario we are decreases. Memory policy attributes importance to past states, less than to the other two categories, but nonetheless taking temporal information into consideration.

Next we have the relationship between the importance of a state entry (e.g., a reservoir level) and the state entry itself. In figure 7.22 we have the average PP score for each reservoir volume and its importance over the stages for all policies.

Interestingly, in the SDDP policy the relationship between the SHAP value of a state and the value of the state itself is similar to all reservoirs and decreases with time. That is, as the stages progress, the policy starts to use other information than the reservoir level itself (i.e. uses the other reservoir levels) to determine the importance of a hydro. Its importance now depends on the other states, while initially it almost only depended on itself. In other words, the impact of a change on the volume of one of the reservoirs depends on the volume of the other reservoirs.

This behavior does not happen in the other policies. In the conditional $[20, 20]$ the PP values are high in different reservoirs and stages. The importance of a reservoirs

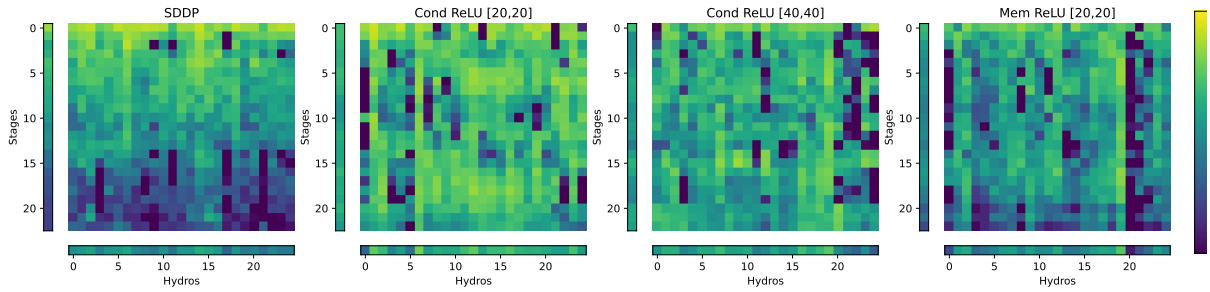


Figure 7.22: Relationship between importance and reservoir levels. Stage means (left) and spatial means (bottom) are shown for all policies.

depends, in general, largely only on the state itself. The conditional policies for the $[40, 40]$ network and $[20, 20]$ with memory have a few reservoirs that consistently have an impact almost independent of the volume itself.

### 7.2.5 Future Cost Function Curvature

Here we will expose the proposed methodology of using the curvature to study a given future cost function. The idea is to show as a proof of concept how the curvature attributes can help in the analysis of the future cost functions. The figure 7.23 illustrates the stages and results of the process, which has already been explained in section 6.2, for the future cost function of the first stage of SDDP: points on the manifold are sampled and these points are projected to a smaller dimension; the future cost value for a whole region is interpolated; the different curvature attributes are calculated in the low dimension; the surface local shape and the anomality of a point are calculated.

When we better study the image 7.23, mainly the local shape chart and the one that shows the clusters of anomalies, we see that many of the so-called anomalous points are in the contour of the shape regions similar to a plane. This behavior change is markedly anomalous compared to the other points.

It becomes of our interest to investigate the reasons of a point to be classified or not as an anomaly. The graphs of figure 7.24 show the SHAP values for the anomaly detection algorithm. We see that some attributes (Dip Angle; Mean Curvature; Gaussian Curvature; Max Curvature; Curvedness) have a much higher response than the others, but not every group has the same behavior. In figure 7.25 we see which attributes had the highest (in absolute value) average importance in the cluster. By their definitions, high values of Curvedness (curvature magnitude), Dip Curvature (curvature in the direction of greater change) and Contour (curvature of the contour line) show a sharp change in

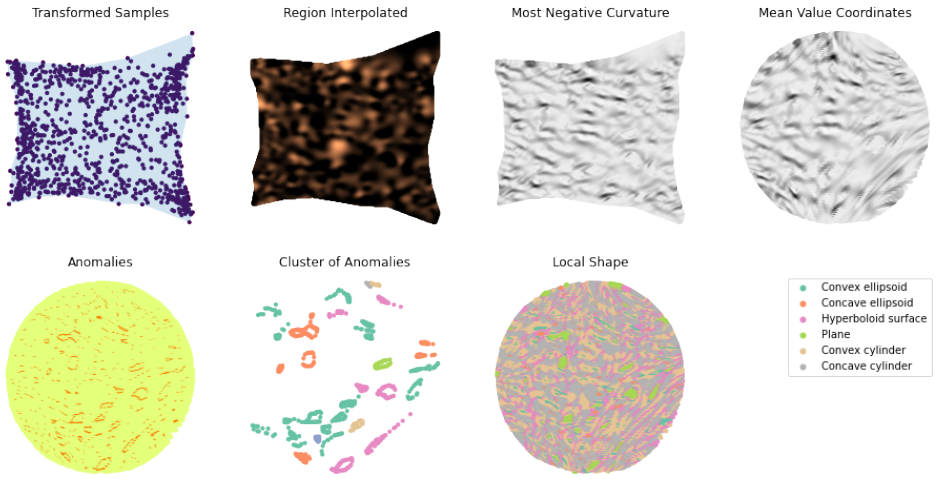


Figure 7.23: The graphs in order: sampled points in the reduced coordinates, with the study region in light blue; the value of the future cost for the entire interpolated region; one of the calculated curvature attributes, the most negative curvature; same attribute but with coordinates in a regular polygon; points classified as anomalous; anomalous points grouped in spatial clusters (colors are repeated); and local surface shape.

at least one direction. The anomalous points are thus, at least on this lower dimensional representation, of deserving our attention.

The group that had Most Positive Curvature as the most important on average seems to be a more heterogeneous group, in which the spatial grouping does not correspond to a similar curvature behavior. We see that some of the points on that group attribute importance to other features such as Dip Curvature and Contour.

For the local shape we can check whether in general they have distinct profiles with respect to the coordinates, that is, with the volumes of the reservoirs. In figure 7.26 we have the profiles of each type of local shape, regardless of space proximity in the reduced manifold. On the left we have the average levels for each reservoirs and on the right the aggregate of all reservoirs. We see that the orders of the reservoirs of the planar regions and of the convex ellipsoid are similar. Both have a average reservoir levels greater than the others. Although the orders are similar, the planar regions have greater variance between the volumes of the individual reservoirs.

The concave, ellipsoid or cylindrical, regions are those we the least total volume and their lower-level reservoirs are those with the highest level in the other two previous regions. They then function as the opposite behavior. The hyperbolic and convex cylindrical regions have more uniform volumes, closer to half. Thus, their total volumes are between the previous two groups.

Low magnitude values can deceive us as to the real local shape. By considering

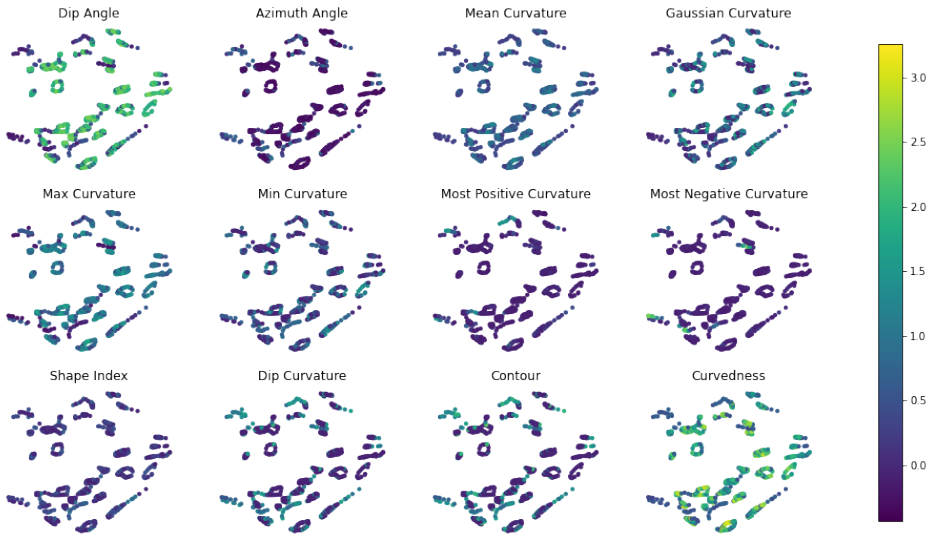


Figure 7.24: SHAP values of the curvature attributes for anomaly detection.


Figure 7.25: Attribute with the greatest average absolute importance for each group.

that values $\{x : |x| \leq \epsilon\}$ are zero for different $\epsilon$, we can study how local shapes change. In figure 7.27 we have the fraction of the interpolated points that are locally hyperboloid for all stages. We see that this behavior is less robust for the initial stages (this hyperboloid behavior is weak), while quite robust for the final stages.

This statistics, the fraction of the interpolated points that are locally hyperboloid, allows us to compare the different models regarding the behavior of the learned functions.

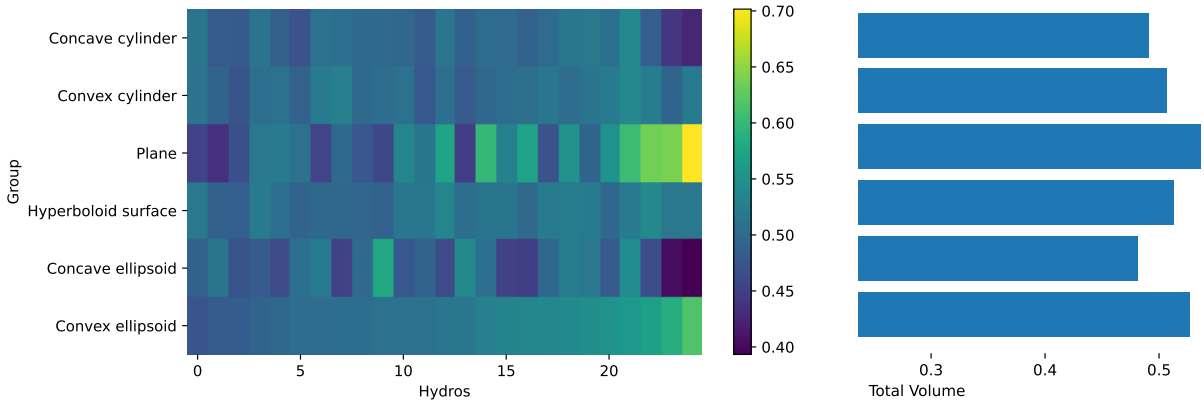


Figure 7.26: Local Shape average profiles. The average volume for the local shapes. left: the average volume of each reservoir with a maximum volume greater than zero. the order of the reservoirs is the order of the average volume on the elliptical convex region. Color represents the fraction of the volume of the reservoir. right: the fraction of the maximum capacity occupied by water for each of the profiles.

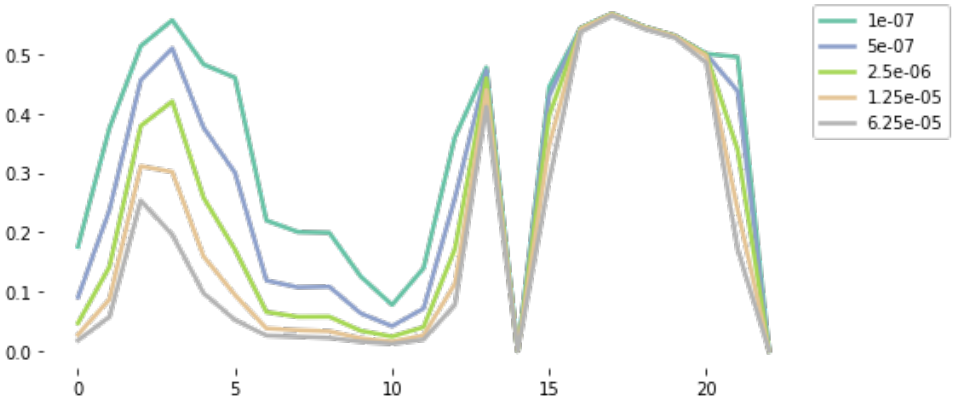


Figure 7.27: Percentage of interpolated points which are in hyperboloid regions for different values of $\epsilon$ over the stages, in which $\{x : |x| \leq \epsilon\}$ are considered zero.

In figure 7.28 we have this comparison. Neural models are much less locally hyperboloid than the policy learned by SDDP.

We can also see how the new coordinates relate to the original coordinates. In figure 7.29 below we have the total volumes set in the new coordinates. As the points were interpolated directly from the reduced region, we can see the original coordinates (the sum at least) of the closest sampled point in the transformed coordinates. That is, the Voronoi diagram [107] of the sampled points.

We see that there is a relationship between the two, in which points in the upper left region have a total volume below the average, and points in the lower right corner have a total volume above the average. So information about the original coordinates are preserved in this lower dimensional representation.

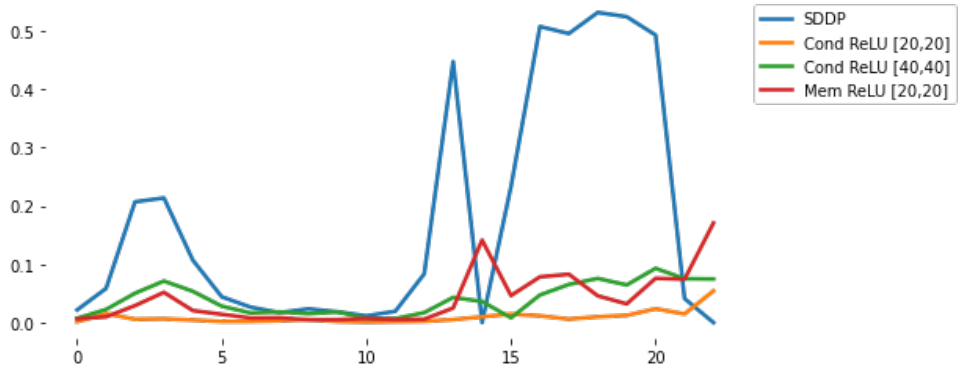


Figure 7.28: Percentage of interpolated points which are in locally hyperboloid regions for different models over the stage, using $\epsilon = 6.25 \cdot 10^{-5}$, in which $\{x : |x| \leq \epsilon\}$ are considered zero.

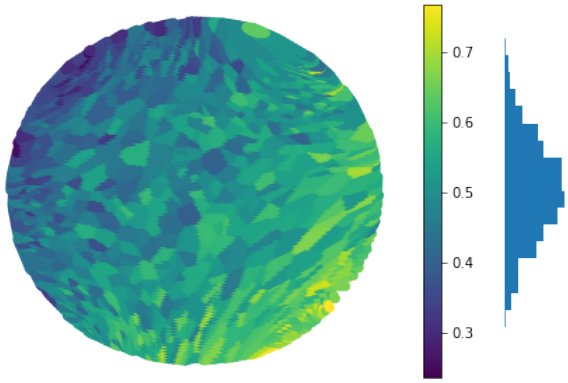


Figure 7.29: Voronoi diagram of the sampled points but on the reduced manifold, colored by the total volume of that point (a proxy for the original coordinates). On the right, the histogram of total volumes.

## 7.3 Optimal Power Flow Problem

In this section we have experiments with synthetic data for the Optimal Power Flow Problem, described in section 4.3. Because it is a problem of non-convex optimization, its computational cost is high and we opted for the use of small instances. The intention is only to show it as proof of concept that the proposed algorithm can be used for this problem. To the extent to what is known by the authors, it is the first time that methods like these are used for non-convex problems that are not MIPs. We first have a discussion on the data used and then two sub-sections with some results.

### 7.3.1 Test Scenarios

The Illinois Center for a Smarter Electric Grid (ICSEG), belonging to The Grainger College of Engineering at The University of Illinois Urbana-Champaign, maintains a repository of publicly available power flow architectures without confidentiality (https://icseg.iti.illinois.edu/power-cases/), including synthetic architectures and architectures commonly used by literature. For example, the architecture used in the work [43], used in our problem modeling, was the IEEE 57-Bus System (https://icseg.iti.illinois.edu/ieee-57-bus-system/).

However, in this work it was chosen to use synthetic instances only as proof of the concept that the algorithm could be used for this problem. Thus, the values and architectures used do not represent real contexts nor necessarily plausible situations and magnitudes.

The demands were simulated in a cosine-like way, adding gaussian noise and adjusting the scale. An example of sample is given in figure 7.30.

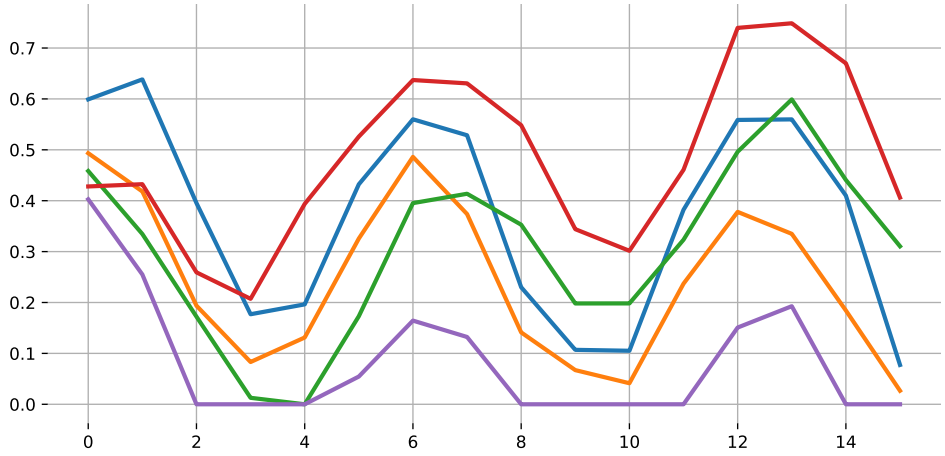


Figure 7.30: Demand Scenario Sample.

As for architectures, two different instances were chosen: a triangular graph ($K_3$); and a random planar graph. We'll now discuss both.

The first instance is simpler. It consists of three buses, all three with generators, with one of them having a demand greater than its own generation capacity and another one having a battery. The third bus acts as a second possible path between the battery and the demand. The architecture is schematized in figure 7.31 below. The cost and constraints constants are described in table 7.1. The admittance matrix $Y = 1/Z$ was built from $Z = R + jX$ in which $X = R = A^T A, A \sim \mathcal{N}(0, 0.1)$.

For the second experiment, the work of [40] was taken as inspiration, in which some strategies of creating random topologies for the graphs are discussed superficially.

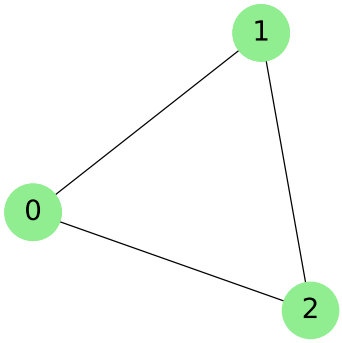


Figure 7.31: Simple Network.

| **Constants** | **Bus 0** | **Bus 1** | **Bus 2** |
|---|---|---|---|
| Constant cost of $P$ | 0 | 0 | 0 |
| Linear cost of $P$ | 1 | 1 | 1 |
| Quadratic cost of $P$ | 2 | 2 | 2 |
| Linear cost of $\Delta P^E$ | 2 | 2 | 2 |
| Quadratic cost of $\Delta P^E$ | 4 | 4 | 4 |
| Linear cost of $\Delta Q^E$ | 2 | 2 | 2 |
| Quadratic cost of $\Delta Q^E$ | 4 | 4 | 4 |
| Battery efficiency $\eta$ | - | 1 | - |
| Minimum battery charge $B^{\min}$ | - | 0 | - |
| Maximum battery charge $B^{\max}$ | - | 2 | - |
| Minimum charging rate $R^{\min}$ | - | -0.5 | - |
| Maximum charging rate $R^{\max}$ | - | 0.5 | - |
| Minimum of real generation $P^{\min}$ | 0 | 0 | 0 |
| Maximum of real generation $P^{\max}$ | 0.5 | 0.5 | 0.5 |
| Minimum of imaginary generation $Q^{\min}$ | 0 | 0 | 0 |
| Maximum of imaginary generation $Q^{\max}$ | 0.5 | 0.5 | 0.5 |
| Minimum of real flow $P_{ij}^{\min} \quad \forall j$ | -0.5 | -0.5 | -0.5 |
| Maximum of real flow $P_{ij}^{\max} \quad \forall j$ | 0.5 | 0.5 | 0.5 |
| Minimum of imaginary flow $Q_{ij}^{\min} \quad \forall j$ | -0.5 | -0.5 | -0.5 |
| Maximum of imaginary flow $Q_{ij}^{\max} \quad \forall j$ | 0.5 | 0.5 | 0.5 |
| Difference without penalization $\Delta P$ | 0.05 | 0.05 | 0.05 |
| Difference without penalization $\Delta Q$ | 0.05 | 0.05 | 0.05 |
| Maximum quadratic flow $S_{ij} \quad \forall j$ | 1 | 1 | 1 |
| Maximum thermal loss $L_{ij} \quad \forall j$ | 0.01 | 0.01 | 0.01 |
| Minimum of real power $\mathrm{Re}(v)$ | 0 | 0 | 0 |
| Maximum of real power $\mathrm{Re}(v)$ | 100 | 100 | 100 |
| Minimum of imaginary power $\mathrm{Re}(v)$ | 0 | 0 | 0 |
| Maximum of imaginary power $\mathrm{Im}(v)$ | 100 | 100 | 100 |

Table 7.1: Constants for the optimal power flow problem.

It is known that the stations do not relate arbitrarily, but are limited to the physical infrastructure, following geographical relationships. A technique that takes into account this geographical relationship is the Delaunay Triangulation [25], which connects vertices

in a neighborhood. This triangulation differs from others by creating triangles whose circumcircles do not contain other points. In this way the triangles are well behaved and connect vertices that are nearby, which is likely in energy systems. In addition, the triangulation generates planar graphs with average degree which does not depend on the size of the network.

The second experiment is thus based on the Delaunay triangulation. The methodology is illustrated in figure 7.32: samples are taken on the plane; the graph is created through the triangulation; edges are drawn to be removed (without disconnecting the graph) in order to reduce the average degree; the vertices are drawn to wether they even have demand and/or batteries.

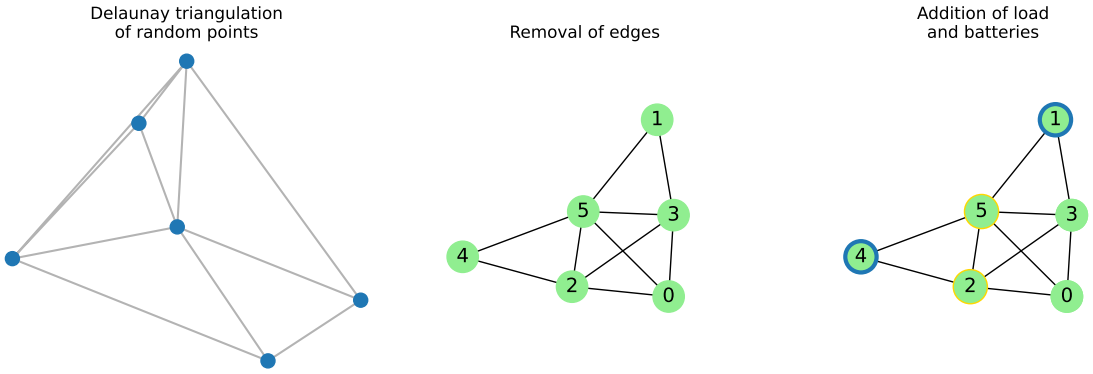


Figure 7.32: Generation of Synthetic Networks.

In the above figure 7.32, the green vertices represent the buses. The yellow border vertices have demand as described above and the blue border ones are batteries. Every node has some generating capacity.

The same constants as the triangular instance were used. The admittance matrix is created in a manner similar to that of the previous instance, but the entries that represent non-edges are zeroed out.

### 7.3.2 Triangular Instance Convergence

In this experiment and in the following one, 5 simulation samples and 5 random samples over the domain were used. The network to the right is a 5-convex manifold with 10 cuts each and was trained by 40 iterations, in the conditional version of the algorithm. After the training, it was possible to estimate lower and higher estimates for the total cost:

| Model | Min | Q1 | Q2 | Q3 | Max |
|---|---|---|---|---|---|
| Greedy | 2.41 | 7.15 | 7.98 | 9.76 | 13.30 |
| SNDP | 2.88 | 5.82 | 6.61 | 8.38 | 11.58 |

Table 7.2: Summary table for the triangular instance total cost.

First, we have the evolution of the loss function in figure 7.33 below. We see that it falls quickly in the first 7 iterations, stabilizing and even growing over the next 25 iterations, to finally fall again. Without the actual future cost function it is difficult to know exactly what's going on in the network during that period.

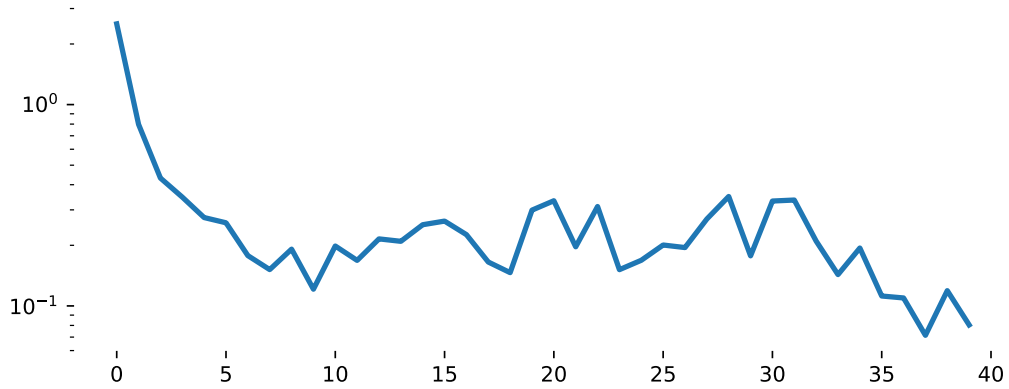


Figure 7.33: Loss function for the Optimum Power Flow problem on the triangular instance.

As a smaller instance, we can investigate more in depth each of the control and state variables and study the dynamics learned. In figure 7.34, we have varied summary graphs of the simulations. The first graph shows the distribution of demand at node 0, cosine-like with increasing variance. The second graph shows the distribution of costs per stage. As we will see below, in the initial state the generators are in their high-generation state of energy, much higher than the initial demand, and therefore there is a greater initial cost. In certain stages, it is even beneficial to have a fast decay, with effective delta greater than zero. We see that this extra energy is smartly stored in the battery, and not simply discarded. From that moment onwards, the cost distribution is similar to the demand, but slightly flattened by the use of the batteries energy. There is though a scenario in which the stored energy is not enough for the flattening and the quadratic cost shows its dangers.

When you check the graphs of the generators, the generator in the battery bus stands out. It only generates the minimum needed while there is still energy on the battery. We see that at the end there is a relative increase, but even the $Q3$ quantile it is around 0.15, relatively low compared to the other generators. As discussed earlier, there are scenarios of greater demand in which the battery is rapidly exhausted and that

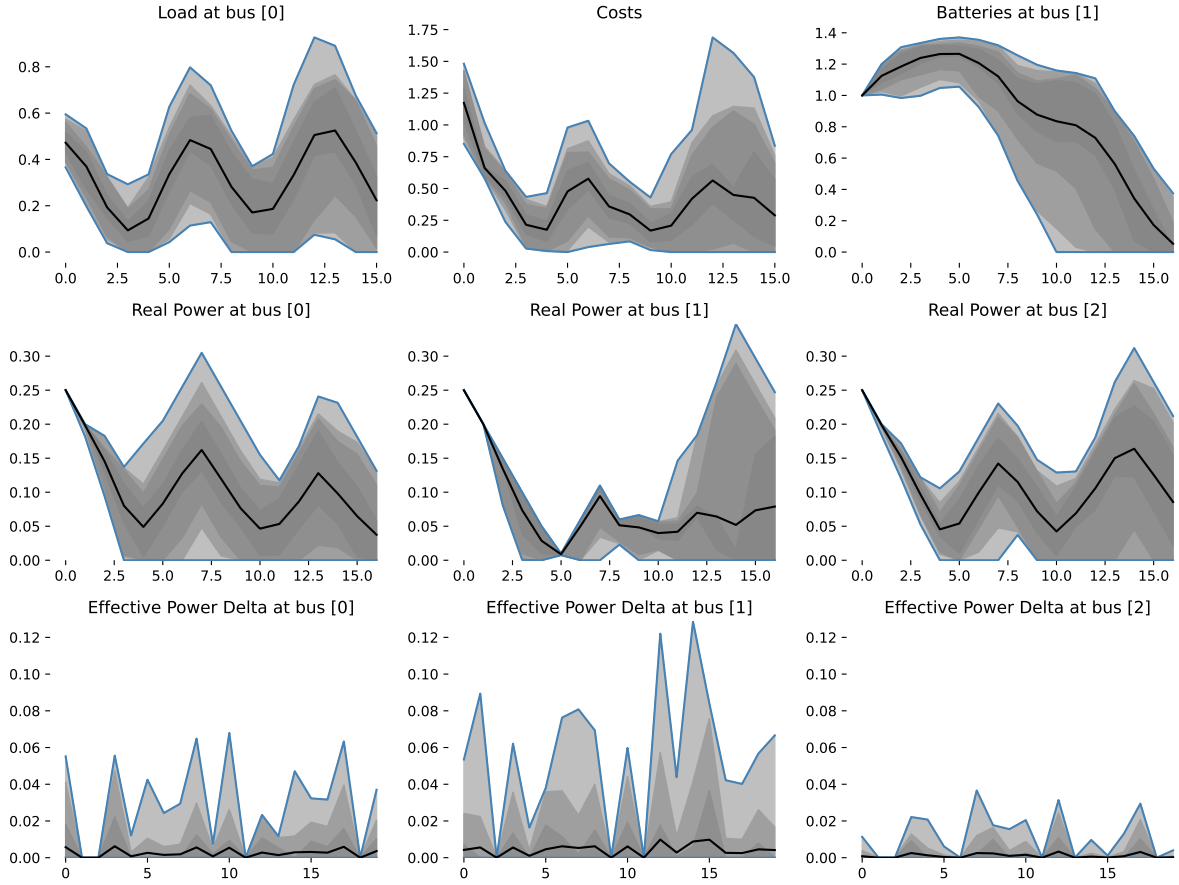


Figure 7.34: Summary graphs for the optimal power flow problem on the triangular instance. The gray regions represent the intervals between the quantiles of $(25\%, 75\%)$, $(10\%, 90\%)$, $(5\%, 95\%)$ and $(\min, \max)$.

generator starts to act like the others, which causes a greater cost. For the effective difference, in the vast majority of scenarios these values are close to zero, with some outlier scenarios.

In the graphs of figure 7.35, we have the power flow between the network nodes. The flows are not symmetrical because we have thermal loss during transmission, but are approximately well similar in absolute value. What we see is a constant flow from buses 1 and 2 to bus 0, while the flow between 1 and 2 can change direction, assuming both negative and positive values. This last flow is smaller in absolute value than the other two. This is a less efficient way of transmission from to energy storage to the demand.

Similarly, we can study the thermal loss of the flows. Note that they are minimal, even when there is the need, in one or another scenario, of an atypical flow. As for the current, which is not part of any constraint of the problem and only arises by the Ohm's Law, it has a large variance.

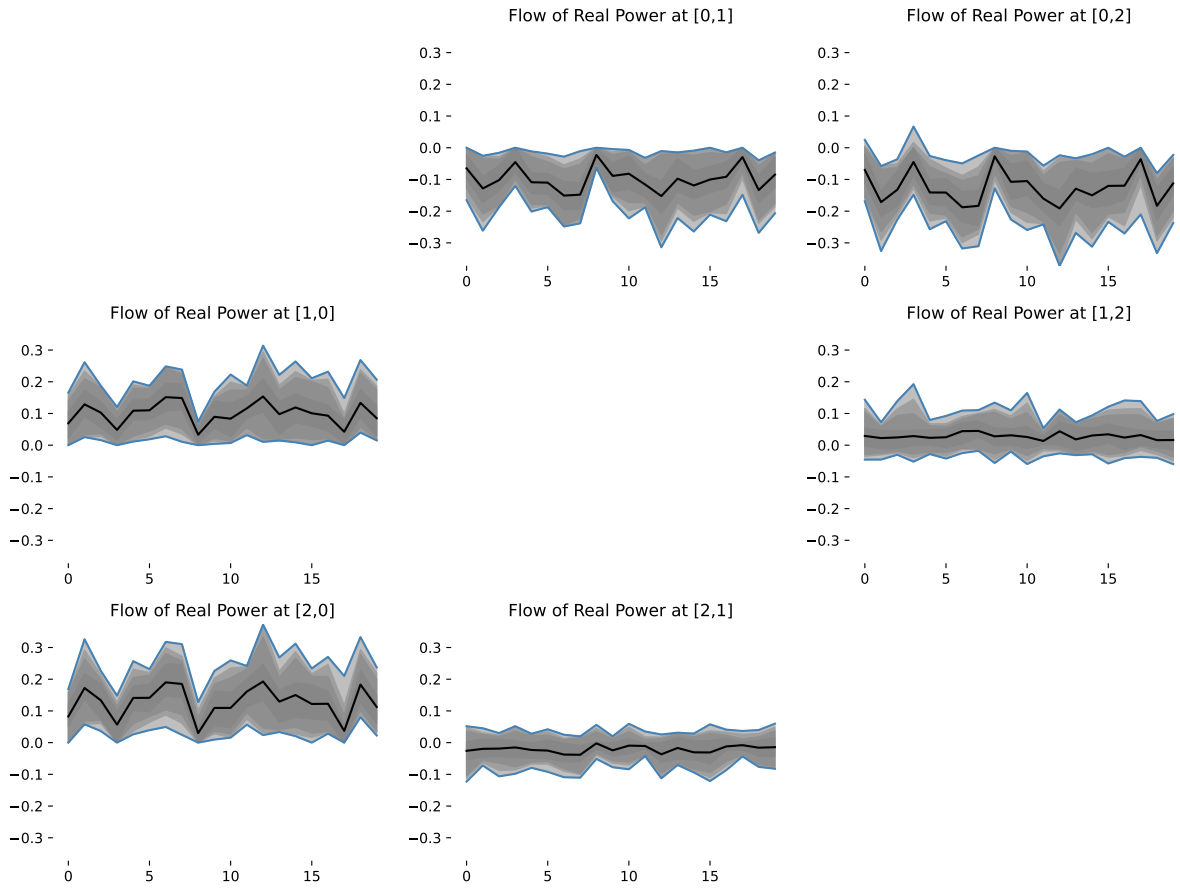


Figure 7.35: Flow graphs for the triangular instance. The gray regions represent the intervals between the quantiles of $(25\%, 75\%)$, $(10\%, 90\%)$, $(5\%, 95\%)$ and $(\min, \max)$.

### 7.3.3 Planar Instance Convergence

The instance tested here is illustrated in the last third of the figure 7.32. The training parameters used were the same as in the previous experiment, with the exception of the number of cuts that was duplicated, given that the state of the problem is twice its size. Thus, the number of parameters to be learned is 4 times greater. Below, in the table 7.3, we have summary statistics for the total costs in the training scenarios. We see that in general the model learned has a lower cost than the greedy strategy, with the exception of the most extreme case, in which the policy learned is not well adjusted. As we see in the figure 7.38, the network seems to not yet have converged, and thus it could be even better. Other strategies could be the use of a larger or more powerful (ReLU netowrk) architecture for the network to the right; to sample states in the regions closer to the ones obtained by simulation this scenario; or to increase the appeareance of this scenario during training.

Because it's a more complex instance, with more buses and connections than the

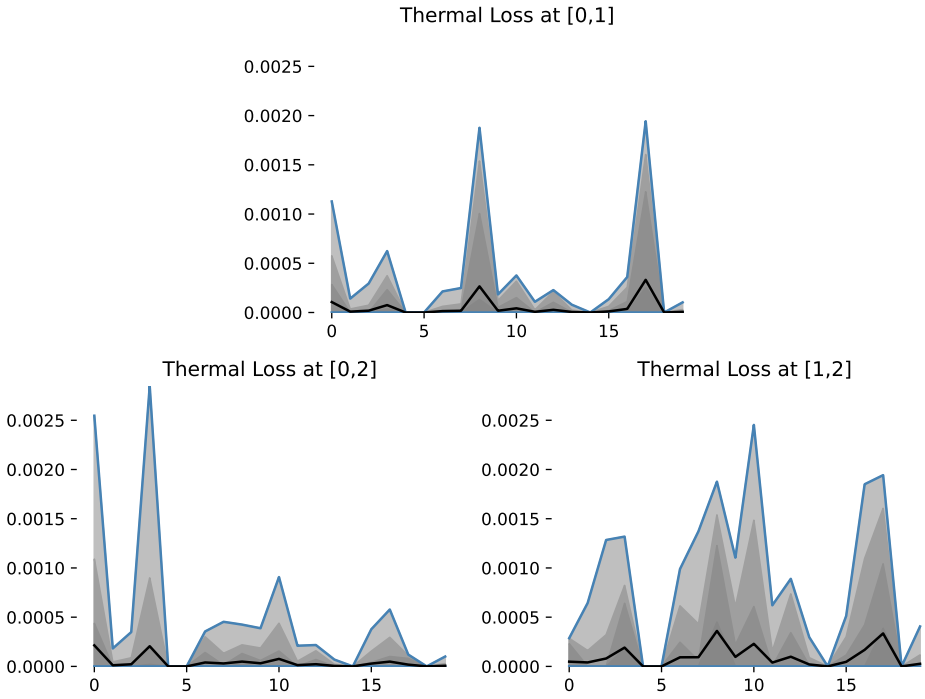


Figure 7.36: Thermal loss graphs for the triangular instance. The gray regions represent the intervals between the quantiles of $(25\%, 75\%)$, $(10\%, 90\%)$, $(5\%, 95\%)$ and $(\min, \max)$.

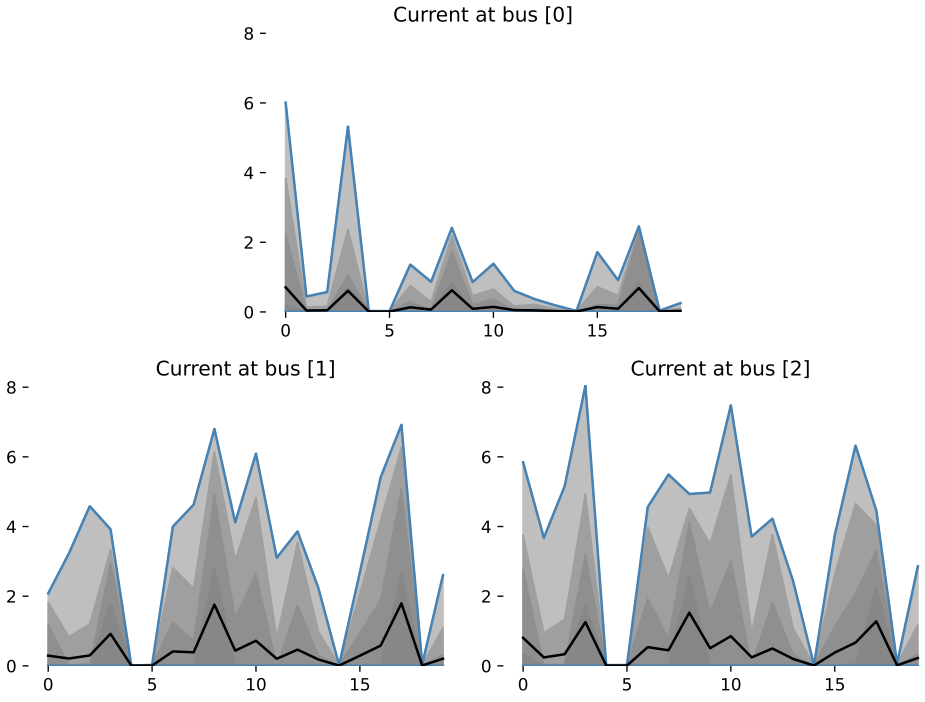


Figure 7.37: Current graphs for the triangular instance. The gray regions represent the intervals between the quantiles of $(25\%, 75\%)$, $(10\%, 90\%)$, $(5\%, 95\%)$ and $(\min, \max)$.

previous, it's more difficult to study each variable individually. In figure 7.39 and 7.40 we have some summary graphs for the greedy strategy and for SNDP.

| Model | Min | Q1 | Q2 | Q3 | Max |
|---|---|---|---|---|---|
| Greedy | 7.2 | 11.39 | 12.42 | 15.65 | 24.49 |
| SNDP | 6.7 | 9.51 | 11.64 | 13.67 | 27.79 |

Table 7.3: Summary table for the planar instance total cost.

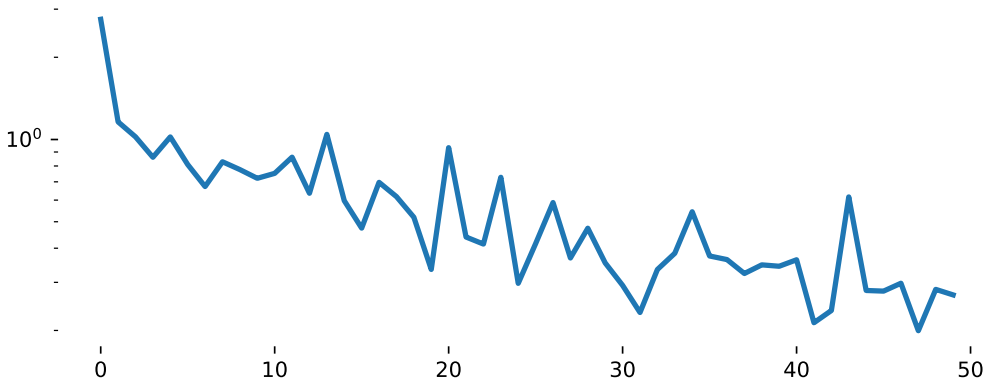


Figure 7.38: Loss function for the Optimum Power Flow problem on the planar instance.

The first difference between the two is in terms of the distribution of costs. The learned policy opts for a slightly higher cost in the early stages due, as we will see, to a faster decrease in the levels of energy generation (initially higher than the demand). Then the costs of the greedy strategy follow a seasonal pattern similar to that of the demand. In the learned policy, this seasonal pattern is also observed, but, with the exception of a more extreme scenario, the costs are lower than that of the greedy policy. The policy learned how to flatten the costs.

In all generators, there is more variance in the early stages in the learned policy than in the greedy one. Generators at the buses 0, 3 and 5 are similar in both policies. Generator 1 in the learned one, with the exception of a discordant scenario, is used with much less intensity. Generator on bus 2 is kept at much lower levels. Generator at bus 4 starts to have a larger variance.

As for the use of batteries, here we have a great difference. The learned policy prefers to save part of the energy to amortise the peaks of seasonal demand, rather than to use in the initial stages that do not have a lot of demand. As the cost of generation is quadratic, flattening the costs has some utility. The battery at bus 1, which has only a direct connection to one of the buses with demand, is less used and can provide power during the two peaks. The other battery, connected to both load nodes (nodes with demand), is emptied at the first peak. Here we can understand the problem of this policy as regarding the extreme case. The two batteries are not able to maintain their levels and finish before the first peak. At the same time, the generators are at lower levels of generation, requiring a large quadratic cost to increase them for the first peak. See bus

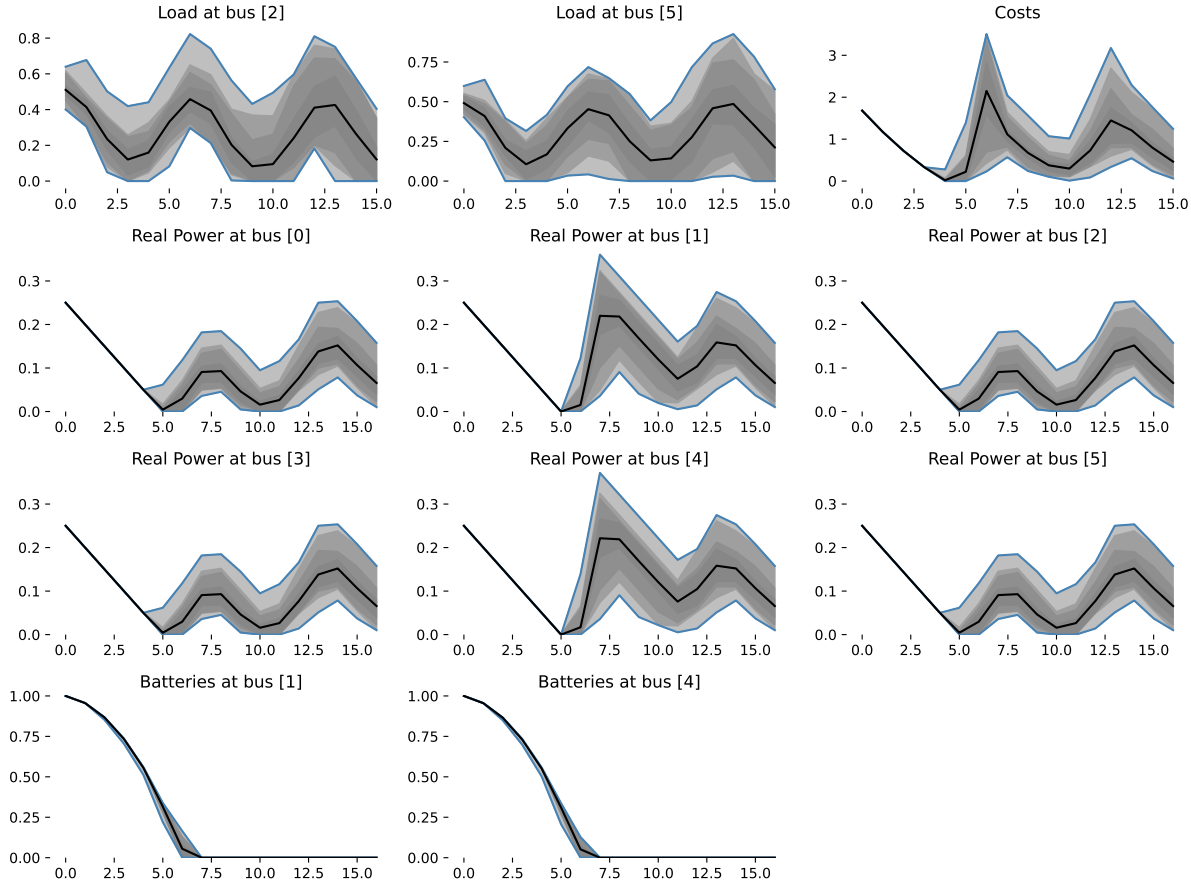


Figure 7.39: Summary graphs for the optimal power flow problem on the planar instance using the greedy strategy. The gray regions represent the intervals between the quantiles of $(25\%, 75\%)$, $(10\%, 90\%)$, $(5\%, 95\%)$ and $(\min, \max)$.

1. As the algorithm is conditioned only on the current observation, it would be possible that, if it knew something about the trend, it learned the battery levels are going to be insufficient and that it is necessary to increase generation earlier.

Following, we have in figure 7.41 the graphs that show the average flow of real power and the average thermal loss. As the thermal loss is symmetrical, only one of the directions is shown. We see that the connections (1,5) and (4,2) have the highest flow, followed by the (4,5), (3,2) and (0,2). The first three represent the batteries powering the demands, while the last two are power reaching bus 2 by sources other than battery in 4. Bus 5 has the connections with the highest thermal loss, possibly by its centrality in the grid.

Finally, we have in figure 7.42 the average current values in the stages for all buses. With the exception of the last two stages, the current values are very small. It is possible to see an increase in the current in the stages of lower demand (2, 3, 8 and 9).

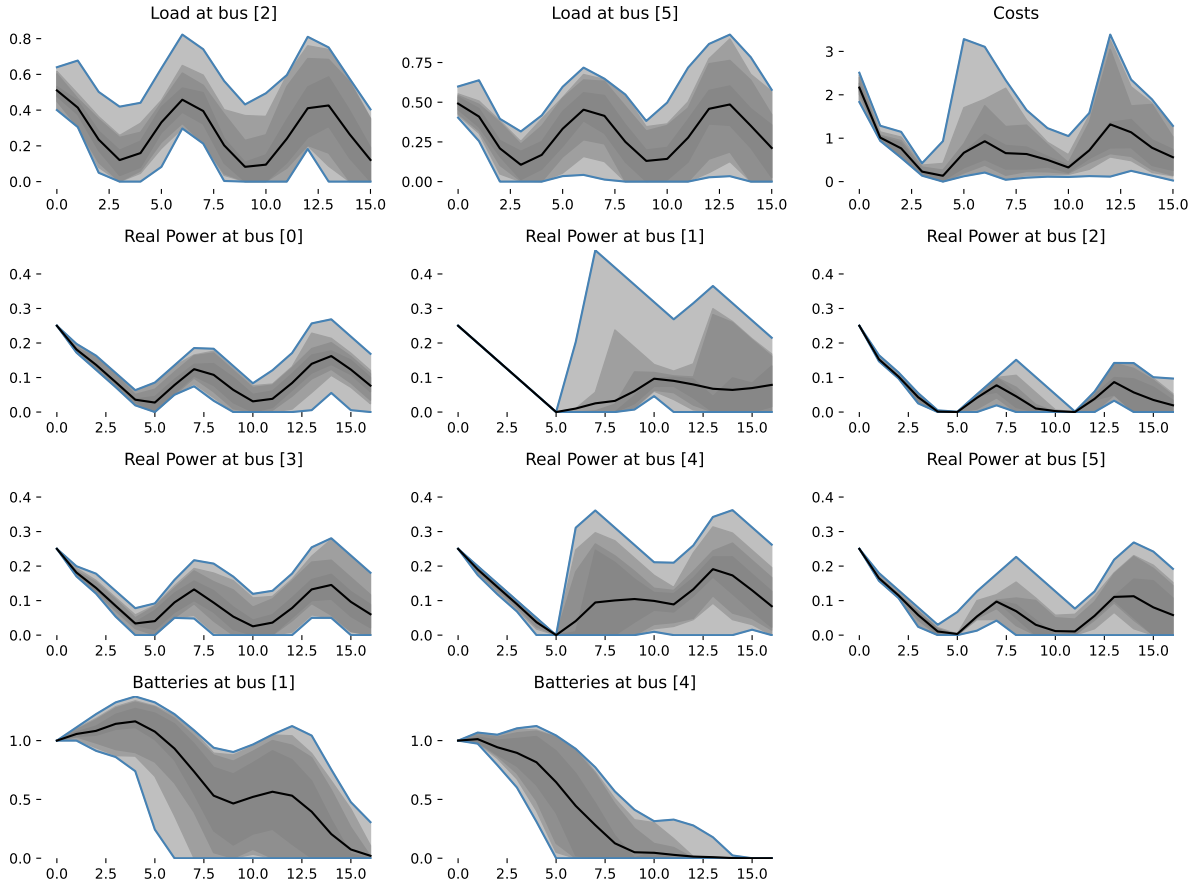


Figure 7.40: Summary graphs for the optimal power flow problem on the planar instance using the SNDP. The gray regions represent the intervals between the quantiles of $(25\%, 75\%)$, $(10\%, 90\%)$, $(5\%, 95\%)$ and $(\min, \max)$.

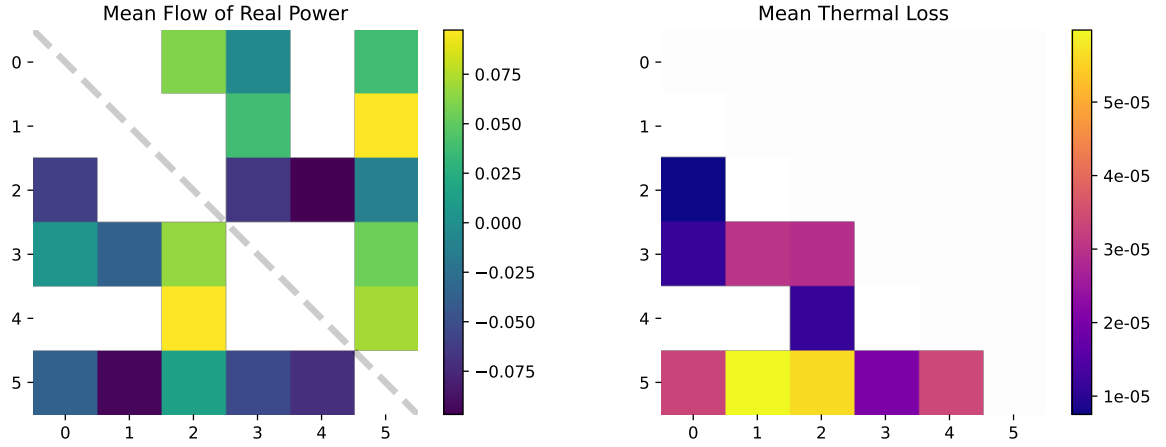


Figure 7.41: Flow and thermal loss graphs for the planar instance.

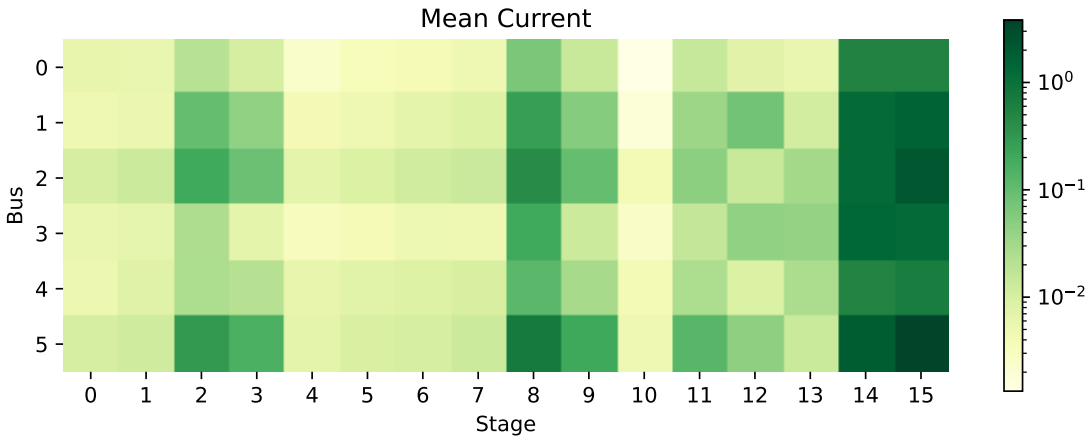


Figure 7.42: Current graphs for the planar instance.

# Chapter 8

# Conclusion and Future Work

In this chapter we have a discussion about what has been achieved throughout the work, as well as possible paths for its continuation.

## 8.1 Conclusion

The work proposed here had two main contributions: a new algorithm to solve MSSO problems, convex or not; and a set of methodologies aimed at understanding the behavior of this and other algorithms regarding the decisions taken.

### 8.1.1 Stochastic Neural Dynamic Programming

As the main object of the work we have the proposal of a new algorithm that seeks to use the prediction power of the neural networks to solve multi-stage stochastic optimization problems, convex or not. Based on the SDDP algorithm, the use of neural networks imposes two main obstacles, the way to model the future cost functions as neural networks during the inference; and the way to train the weights of the network, since the external solvers of the subproblems are not differentiable. For the first, an integer programming model was used and incorporated to the subproblem. For the second, a new loss function was proposed which allows the algorithm to work similarly to SDDP, to iteratively modify the future cost function. By experimenting the algorithm in three different problems, we see that the proposed technique was capable of training the network weights.

In the experiments of the control problem in one dimension we see that the network approximates well the actual future cost function. As for the dispatch problem, which we

know is convex, the algorithm cannot reach the SDDP level, having an average total cost about 4% greater. As for the problem of optimal power flow, there is no use in literature of these algorithms for this non-convex problem. But we see that the learned policy is superior to the naive one.

A limitation of the proposed algorithm is that it can be computationally expensive, especially for problems with many decision variables and stages, since the iterative process of improving the network requires solving a considerable number of MILP (or MIQP) subproblems. Unlike SDDP that uses convex cuts, the networks on the right are integer programming and the increase in their capacity implies a greater computational cost.

The use of neural networks allows us to go beyond optimizing only the average (or other expected risk measurement), as we can condition our predictions on the observations, in latent information or even in temporal trends (hidden states). There's no longer the need to limit ourselves to probabilistic independence between the stages uncertainty.

### 8.1.2 Future Cost Function Explanations

In addition to solving the proposed problems, being long-term applications in the real world, it is of interest to understand the reasons why a certain decision is preferable, as well as the risks it can bring. This is not a trivial task and its objectives are not always clear. This work proposes a set of methodologies that can help in this analysis, but there is no single way of using them. The approaches work with the use of two main techniques, SHAP and curvature of surfaces. All the methodology has been applied to the problem of hydrothermal dispatch for the Colombian System.

The use of SHAP allows us to understand two things: how the reservoirs and other variables (observations, hidden states) affect the future cost of a decision; how the reservoirs interact with each other. As for the first, we can see how the different models learn quite different policies by not giving the same importance to the reservoirs over time. In a situation in the real world, it is not always possible to replicate the decision of the algorithm, so it is important for the operator to know what are the critical points of the system for the current model. As for the second part, we see that the models also differ on the relationship between the reservoirs, while SDDP is practically taking independent decisions at the beginning, the neural models have dependency since the beginning.

The idea of using curvature metrics for the future cost function arises when migrating the policy learned to the real world. In real applications, it is not always possible to accurately replicate the best decision of the algorithm. However, we don't know if the neighborhood of the best decision has a similar behavior or if there is a sudden change

with great long-term impacts. To quantify this, we can see how the function behaves locally. The use of curvature opens space for a range of possible analyses of that local shape.

## 8.2 Future Work

Here we have a brief discussion on possible research paths to enhance the algorithm. The paths are 4: when do we have a convergence guarantee; what are possible performance improvements both at the algorithm level and at the implementation level; how can we use the insights of the curvature method during inference; and about other applications that the method could be applied to.

### 8.2.1 Convergence Guarantees

This work does not deal with convergence guarantees, only delineating an inductive argument in the section 5.1. Thus, it would be of our interest that it does not support itself only by experiments, since the decisions taken by it possibly have great financial, environmental etc. impacts.

The objective would be to show that for a given problem and architecture of network to the right the model converges in a finite amount of steps to the surroundings (a $\epsilon$ error) of the optimal policy. The better the architecture and the easier the problem, the lower would be the surrounding region when converging. It is possible that an architecture is not able to represent the future cost function, even if conditioned to the observations. In this case, the model would not converge. Therefore, there would be questions if the algorithm always converges, and if not, when does it converge.

Unlike other MSSO algorithms, which by construction create lower bounds of the actual future cost function, SNDP algorithm has no type of theoretical technique, only through simulation. Another question would then to study mechanisms that estimate lower or higher bounds of the future cost function in a non-simulated way. It is not clear at the time which would be these mechanisms.

### 8.2.2 Computational and Algorithmic Convergence

The performance gains are divided into algorithmic changes and implementation changes. In the first we still have another division, of how to modify the current flow of the algorithm and using the subproblem solution data in a more efficient way; and that of using other theoretical constraints that could ensure better efficiency.

The first question then would be about the best way to use the subproblem solutions. Being the most expensive part of the algorithm, would it be possible to use a smaller number of solutions steps? A strategy on this path is about the penultimate stage, i.e. the last future cost function, in which each subproblem sample the actual future cost, not an estimate. It could make sense to keep all these samples over the iterations to strengthen the learning of this specific future cost function, in the hope that learning propagates to the other stages.

A theoretical constraint would be on the networks to the right. This work used the modeling 3.8 to represent the ReLU networks within the subproblems. Other modelings exist. How do they compare to each other regarding their relaxations? There is a modeling that its relaxation is always closer to the solution compared to the others? In other words, a more compact one? In addition, throughout the experiments we treated the architectures arbitrarily, only observing their predictive performance. Are there architectures that, for a given modeling, their relaxations are always closer to the integral solution? By some symmetry, the number of layers and the number of neurons per layer could affect the relaxation of the network. Knowing that it is possible to model matrix convolution as a matrix multiplication [78], it could be possible that a convolutional network has a relaxation more or less tighter to the solution. Of course the represented state has to possess some spatial relationship. Another architectural change is the use of rigid attention [111]. Is its relaxation compact? We could still compare sparse networks and verify if there is any gain. Many such questions can be studied.

We saw that neural networks can model an exponentially larger number of linear regions. There are several integer programming problems whose modeling with an exponential number of variables and constraints through the use of lazy constraints have better experimental performance than polynomial modelings because their relaxations are more compact. Would it be possible to list all the linear regions of a ReLU neural network and model it with lazy constraints?

There are also theoretical improvements that can be studied over the course of a training. As the training advances, the most explored region of the state space will have its future cost with greater resolution than other regions. How does this affect the optimization of a ReLU network? Is it more or less compact?

On the convergence of the weights for the network to the right, recent work [82]

proposes a reparametrization of the output network weights that experimentally shows more stable and fast convergence. The proposed modifications do not cause an increase in cost in training or in inference and therefore show themselves as a simple modification to the algorithm.

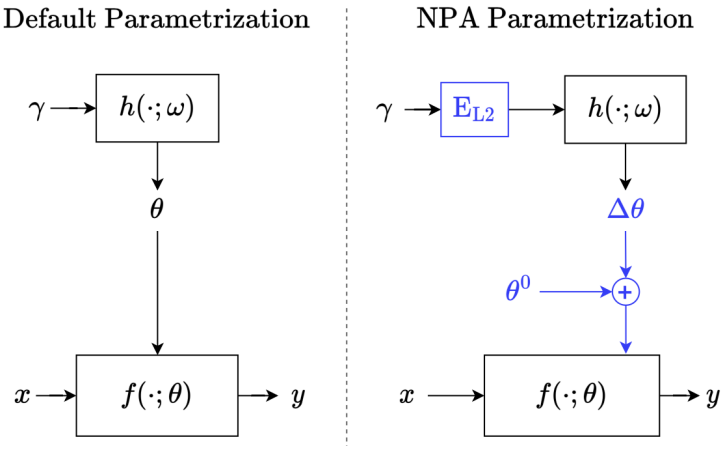


Figure 8.1: With the NPA reparametrization, the hyper-net input $\gamma$ is first projected into a vector space of constant norm and its predictions $\Delta\theta$ are summed to a set of learned parameters $\theta^0$ independent of the input $\gamma$. Figure by [82].

Finally, we have possible implementations improvements. In the algorithms described in section 5.1, it is noted that the loop over the samples the samples can be easily parallelized. But deeper changes can be tested. In the article for SDDP [86], the authors instead of solving the subproblems as linear programs, they use an algorithm inspired by the problem of maximum flow in networks whose performance is better. So, it would be interesting if the modeling here proposed could be incorporated into the specific algorithm of a subproblem. It doesn't seem trivial to minimize the non-convex networks efficiently, but the manifolds of $k$-convex for small $k$ can be useful. In general, one can still experiment other solvers besides integer programming ones. A new neural method [33] can solve the traditional Optimum Power Flow problem (single stage, no batteries) in a tenth of the time specific solver for the problem require, which are already faster than the integer quadratic programming solvers used in this work. As it also require a training step, one may wonder how to combine the two algorithms so that both train at the same time.

### 8.2.3 Curvature

In this work, curvature was used to study the local shape and the presence or not of anomalies on the surfaces of the future cost functions. That is, an analysis *post hoc* of the model. But would it be possible to introduce part of the analysis back to the decision making? See image 8.2.

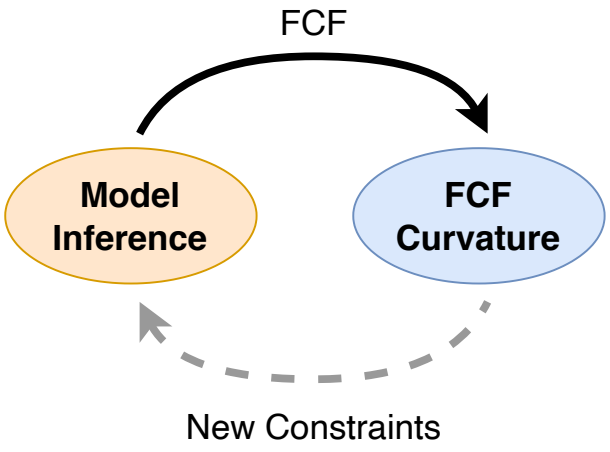


Figure 8.2: It is possible to incorporate constraints into the original model based on what is observed in the curvature plots.

If we are able to identify unwanted regions in the state space, hyperboloid or anomalous, we can introduce to the model during inference (model will no longer be trained) new constraints that force the optimal choice to penalize or even avoid these regions.

In [97], the authors study optimization problems in a geometric context. Some of the problems can be incorporated as quadratic constraints or penalties to our sub-problems. See for example figure 8.3. It illustrates the problem of finding the minimum distance between two polytopes. If we know how to represent the regions to be avoided as polytopes, we can add a quadratic penalty or constraint for this distance distance.

To minimize the distance between polytopes we have:

$$\begin{aligned} \min_{\mathbf{x}\in\mathbb{R}^n} \quad & x^T C^T C x \\ & \sum_{i=1}^{r} x_i = 1, \\ & \sum_{i=1}^{n} x_i = 1, \\ & x \geq 0 \end{aligned} \tag{8.1}$$

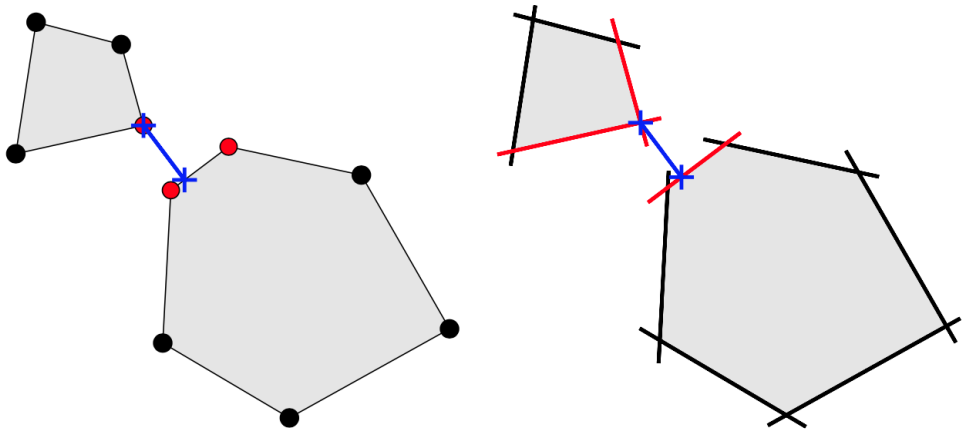

Figure 8.3: Polytope distance (primal and dual version). Figure by [97].

It may be that decision makers do not want to remove the solution from a problematic region, but rather to bring it to a well behaved region. One can then minimize the smallest ball, ring or ellipse that contains both the solution and the points on the area of interest. Figure 8.4 illustrates the three problems

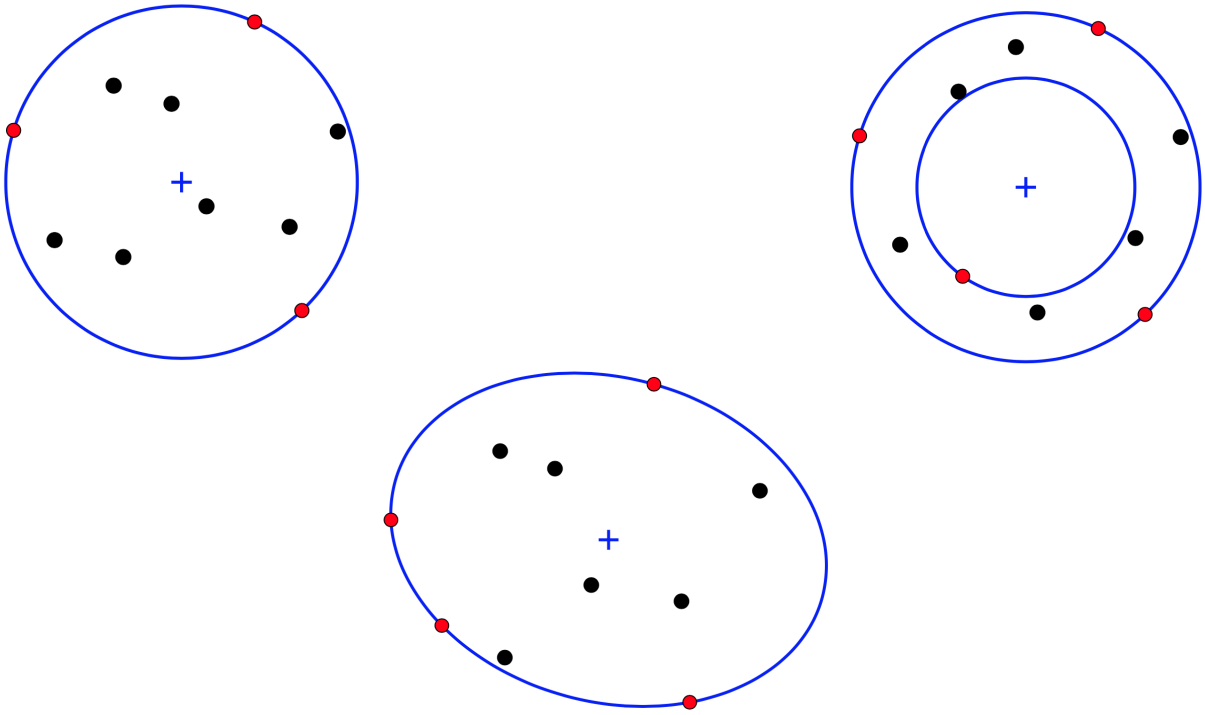

Figure 8.4: Geometric Optimization Problems. (Top, Left) Smallest enclosing ball. (Top, Right) Smallest enclosing Annulus. (Bottom, center) Smallest enclosing ellipse. Figures by [97].

We have, respectively, the problems of the smallest enclosing ball 8.2, annulus 8.3 and ellipse 8.4.

$$\begin{aligned} \min_{\mathbf{x}\in\mathbb{R}^n} \quad & x^T C^T C x - \sum_{i=1}^{n} p_i^T p_i x_i \\ & \sum_{i=1}^{n} x_i = 1, \\ & x \geq 0, \end{aligned} \tag{8.2}$$

in which $P = \{p_1, ..., p_n\}$ is the set of points and $C$ é a the matrix $(p_1, ..., p_n)$. The center of the smallest enclosing ball is given by $\sum_{i=1}^{n} p_i x_i^*$ and its radius is the square root of the minimum cost in $x^*$.

$$\begin{aligned} \min_{\mathbf{x}\in\mathbb{R}^n} \quad & \beta - \alpha \\ & \sum_{j=1}^{d} 2p_j^i c_j \leq p_i^T p_i - \alpha, && i = 1, ..., n \\ & \sum_{j=1}^{d} 2p_j^i c_j \geq p_i^T p_i - \beta && i = 1, ..., n \end{aligned} \tag{8.3}$$

where $P = \{p_1, ..., p_n\}$ is the set of points. The radii are given by $r^2 = \alpha^* + \|c^*\|^2$ and $R^2 = \beta^* + \|c^*\|^2$.

$$\begin{aligned} \min_{\mathbf{x}\in\mathbb{R}^n} \quad & -\log\det(M) \\ & (p_i - c)^T M (p_i - c) \leq 1, && i = 1, ..., n \\ & M \text{ is positive definite,} \end{aligned} \tag{8.4}$$

in which $P = \{p_1, ..., p_n\}$ is the set of points.

These modelings are, respectively, a quadratic program, a linear program and a convex program. The latter, when in two dimensions, reduces to a quadratic programming.

### 8.2.4 Other Applications

Finally, we can investigate how the algorithm proposed here can be used for other applications other than MSSO problems.

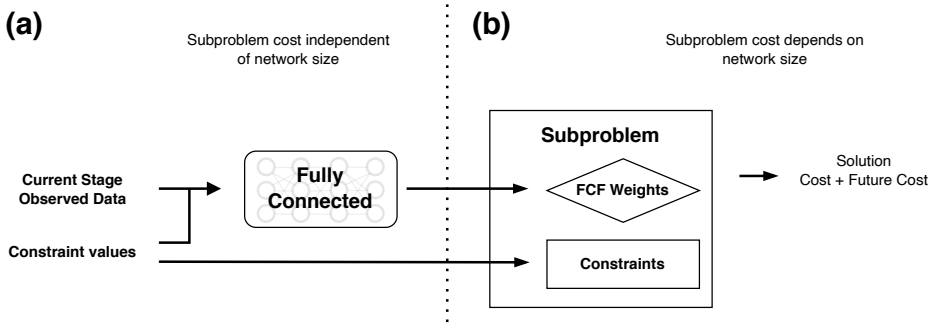


Figure 8.5: SNDP Schema conditionated on different problem setups.

One first application would be about the impacts on the future cost of a change in the infrastructure of a problem, for example the construction of a new reservoir or the connection of two buses in the network. A simple strategy is to train two networks, before and after the modification, and check how the cost changes. For the study of multiple simultaneous modifications this strategy does not scale very well. But it would be possible to train a single network to the left for multiple subproblems with different settings, in which the settings themselves are part of the input of the network. So we could train a model who knows how to find the future cost functions conditioned on the infrastructure. The step of searching for the ideal configuration would only demand the inference on this single model. See figure 8.5.

In this work we dealtproblems with a finite, known and constant number of stages. However, there are optimizations of infinite horizons or finite horizons with a stochastic number of stages. There is a varied collection of games whose actions can be modeled as integer programs, for example Rummikub [26]. It is interesting to investigate how to adapt the algorithm to these cases.

If we discard the time component as a whole, it would be possible to use the algorithm strategy to solve problems such as those described in the sub-section 2.2.2, in which we have a neural network whose output is an optimization solution. The network to the right would predict the loss function of the optimization solution, in a completely differentiable way. Depending on which subproblem it is concerned, the use of convex networks or $k$-convex would be enough to depict the function.

# Bibliography

[1] Thomas Ackermann. Wind power in power systems. 2005.

[2] Akshay Agrawal, Brandon Amos, Shane Barratt, Stephen Boyd, Steven Diamond, and J. Zico Kolter. Differentiable convex optimization layers. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[3] Shabbir Ahmed, Filipe Goulart Cabral, and Bernardo Freitas Paulo da Costa. Stochastic lipschitz dynamic programming. *Mathematical Programming*, 191(2):755–793, 2022.

[4] Erik J Amézquita, Michelle Y Quigley, Tim Ophelders, Elizabeth Munch, and Daniel H Chitwood. The shape of things to come: Topological data analysis and biology, from molecules to organisms. *Developmental Dynamics*, 249(7):816–833, 2020.

[5] Alberto Archetti, Marco Cannici, and Matteo Matteucci. Neural weighted a*: Learning graph costs and heuristics with differentiable anytime a*. In Giuseppe Nicosia, Varun Ojha, Emanuele La Malfa, Gabriele La Malfa, Giorgio Jansen, Panos M. Pardalos, Giovanni Giuffrida, and Renato Umeton, editors, *Machine Learning, Optimization, and Data Science*, pages 596–610, Cham, 2022. Springer International Publishing.

[6] Philippe Artzner, Freddy Delbaen, Jean-Marc Eber, and David Heath. Coherent measures of risk. *Mathematical finance*, 9(3):203–228, 1999.

[7] Elbio Avanzini. *Multistage Stochastic Programming as Flexibility Source in Highly Uncertain Environments: Its Value in an Agriculture Application.* PhD thesis, Pontificia Universidad Catolica de Chile (Chile), 2022.

[8] Jongmin Baek, Anand Deopurkar, and Katherine Redfield. Finding geodesics on surfaces. *Dept. Comput. Sci., Stanford Univ., Stanford, CA, USA, Tech. Rep*, 2007.

[9] John P Barton and David G Infield. Energy storage and its use with intermittent renewable energy. *IEEE transactions on energy conversion*, 19(2):441–448, 2004.

[10] Richard Bellman. The theory of dynamic programming. *Bulletin of the American Mathematical Society*, 60(6):503–515, 1954.

[11] Richard Bellman. A markovian decision process. *Journal of mathematics and mechanics*, pages 679–684, 1957.

[12] Daniel Bienstock. Practical solution approaches to optimization and engineering: Case studies in mine planning and electrical power. 2022.

[13] J. Birge. Decomposition and partitioning methods for multistage stochastic linear programs. *Oper. Res.*, 33(5):989–1007, 1985.

[14] Garrett Birkhoff. Integration of functions with values in a banach space. *Transactions of the American Mathematical Society*, 38(2):357–378, 1935.

[15] A. Brigatto, A. Street, and D. Valladao. Assessing the cost of time-inconsistent operation policies in hydrothermal power systems. *IEEE Transactions on Power Systems*, 32(6):4541–4550, 2017.

[16] Viktram S Budhraja, Fred Mobasheri, Margaret Cheng, Jim Dyer, Eduyng Castaño, Stephen Hess, and J Eto. California's electricity generation and transmission interconnection needs under alternative scenarios. *California Energy Commission, Tech. Rep*, 2004.

[17] J Carpentier. Contribution to the economic dispatch problem. *Bulletin de la Societe Francoise des Electriciens*, 3(8):431–447, 1962.

[18] Juan Manuel Carrasco, Leopoldo Garcia Franquelo, Jan T Bialasiewicz, Eduardo Galván, Ramón Carlos PortilloGuisado, MA Martin Prats, José Ignacio León, and Narciso Moreno-Alfonso. Power-electronic systems for the grid integration of renewable energy sources: A survey. *IEEE Transactions on industrial electronics*, 53(4):1002–1016, 2006.

[19] Lawrence Cayton. Algorithms for manifold learning. *Univ. of California at San Diego Tech. Rep*, 12(1-17):1, 2005.

[20] S. Cerisola, J. Latorre, and A. Ramos. Stochastic dual dynamic programming applied to nonconvex hydrothermal models. *European Journal of Operational Research*, 218(3):687–697, 2012.

[21] W. Chandler, A. Glimn, P. Dandeno, and L. Kirchmayer. Short-range economic operation of a combined thermal and hydroelectric power system. *Trans. AIEE (Power Apparatus and Systems)*, 72:1057–1065, 1953.

[22] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

[23] E. Chowdhury and S. Rahrnan. A review of recent advances in economic dispatch. *IEEE Transactions on Power Systems*, 5(4):1248–1259, 1990.

[24] H. Dai, Y. Xue, Z. Syed, D. Schuurmans, and B. Dai. Neural stochastic dual dynamic programming. *CoRR*, abs/2112.00874, 2021.

[25] Boris Delaunay. Sur la sphère vide. a la mémoire de georges voronoi (on the empty sphere. in memory of georges voronoï). *Bulletin de l'Académie des Sciences de l'URSS. Classe des sciences mathématiques et naturelles*, 6:793, 1934.

[26] Dick Den Hertog and PB Hulshof. Solving rummikub problems by integer linear programming. *The Computer Journal*, 49(6):665–669, 2006.

[27] Boyang Deng, Kyle Genova, Soroosh Yazdani, Sofien Bouaziz, Geoffrey Hinton, and Andrea Tagliasacchi. Cvxnet: Learnable convex decomposition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 31–44, 2020.

[28] Ulrich Derigs. *The Min-Cost Perfect Matching Problem*, pages 200–253. Springer Berlin Heidelberg, Berlin, Heidelberg, 1988.

[29] V. Souza Dias, M. Pereira da Luz, G. Medero, and D. Nascimento. An overview of hydropower reservoirs in brazil: Current situation, future perspectives and impacts of climate change. *Water*, 10:592–610, 2018.

[30] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[31] A. Diniz, F. Costa, M. Maceira, T. Norbiato, L. Santos, and R. Cabral. Short/mid-term hydrothermal dispatch and spot pricing for large-scale systems-the case of brazil. In *Power Systems Computation Conf. (PSCC)*, page 1–7, 2018.

[32] A. Diniz, M. Maceira, C. Vasconcellos, and D. Penna. A combined sddp/benders decomposition approach with a risk-averse surface concept for reservoir operation in long term power generation planning. *Ann. Oper. Res.*, 292(2):649–681, 2020.

[33] Priya L Donti, David Rolnick, and J Zico Kolter. Dc3: A learning method for optimization with hard constraints. *arXiv preprint arXiv:2104.12225*, 2021.

[34] Oscar Dowson, Andy Philpott, Andrew Mason, and Anthony Downward. A multi-stage stochastic optimization model of a pastoral dairy farm. *European Journal of Operational Research*, 274(3):1077–1089, 2019.

[35] Herbert Edelsbrunner, David Kirkpatrick, and Raimund Seidel. On the shape of a set of points in the plane. *IEEE Transactions on information theory*, 29(4):551–559, 1983.

[36] Merrill M. Flood. The traveling-salesman problem. *Operations Research*, 4(1):61–75, 1956.

[37] Patrick J Flynn and Anil K Jain. On reliable curvature estimation. In *CVPR*, volume 88, pages 5–9. Citeseer, 1989.

[38] Kunihiko Fukushima. Cognitron: A self-organizing multilayered neural network. *Biological Cybernetics*, 20(3):121–136, Sep 1975.

[39] Carl Friedrich Gauss. *Disquisitiones Generales Circa Superficies Curvas (General Investigation into Curved Surfaces)*, volume 1. Typis Dieterichianis, 1827.

[40] Kathleen M Gegner, Adam B Birchfield, Ti Xu, Komal S Shetye, and Thomas J Overbye. A methodology for the creation of geographically realistic synthetic power flow models. In *2016 IEEE Power and Energy Conference at Illinois (PECI)*, pages 1–6. IEEE, 2016.

[41] A Geoffrion. Lagrangian relaxation for integer programming. *Mathematical Programming Study*, 2:82–114, 1974.

[42] Ajit Gopalakrishnan, Arvind U Raghunathan, Daniel Nikovski, and Lorenz T Biegler. Global optimization of optimal power flow using a branch & bound algorithm. In *2012 50th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 609–616. IEEE, 2012.

[43] Ajit Gopalakrishnan, Arvind U Raghunathan, Daniel Nikovski, and Lorenz T Biegler. Global optimization of multi-period optimal power flow. In *2013 American Control Conference*, pages 1157–1164. IEEE, 2013.

[44] John C Gower. Some distance properties of latent root and vector methods used in multivariate analysis. *Biometrika*, 53(3-4):325–338, 1966.

[45] Michael Grant, Stephen Boyd, and Yinyu Ye. *Disciplined Convex Programming*, pages 155–210. Springer US, Boston, MA, 2006.

[46] Alfred Gray. Modern differential geometry of curves and surfaces with mathematica, 1997.

[47] Alwin Haensel, Michael Mederer, and Henning Schmidt. Revenue management in the car rental industry: A stochastic programming approach. *Journal of Revenue and Pricing Management*, 11:99–108, 2012.

[48] Lars Hellemo, Kjetil Midthun, Asgeir Tomasgard, and Adrian Werner. Multi-stage stochastic programming for natural gas infrastructure design with a production perspective. In *Stochastic Programming: Applications in Finance, Energy, Planning and Logistics*, pages 259–288. World Scientific, 2013.

[49] A. Helseth, M. Fodstad, and B. Mo. Optimal medium-term hydropower scheduling considering energy and reserve capacity markets. *IEEE Trans. Sustain. Energy*, 7(3):934–942, 2016.

[50] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus), 2020.

[51] Felix Hensel, Michael Moor, and Bastian Rieck. A survey of topological machine learning methods. *Frontiers in Artificial Intelligence*, 4, 2021.

[52] David Hilbert and Stephan Cohn-Vossen. Anschauliche geometrie. 1932.

[53] Shaolong Hu, Chuanfeng Han, Zhijie Sasha Dong, and Lingpeng Meng. A multi-stage stochastic programming model for relief distribution considering the state of road network. *Transportation Research Part B: Methodological*, 123:64–87, 2019.

[54] Di Huang, Jiping Xing, Zhiyuan Liu, and Qinhe An. A multi-stage stochastic optimization approach to the stop-skipping and bus lane reservation schemes. *Transportmetrica A: Transport Science*, 17(4):1272–1304, 2021.

[55] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

[56] Sungil Kim and Heeyoung Kim. A new metric of absolute percentage error for intermittent demand forecasts. *International Journal of Forecasting*, 32(3):669–679, 2016.

[57] D. Kingma and J. Ba. Adam: A method for stochastic optimization. In *Intl Conf. on Learning Representations, ICLR*, 2015.

[58] Studiosus Kirchhoff. Ueber den durchgang eines elektrischen stromes durch eine ebene, insbesondere durch eine kreisförmige (on the passage of an electric current through a plane, especially a circular one). *Annalen der Physik*, 140(4):497–514, 1845.

[59] Jan J Koenderink and Andrea J Van Doorn. Surface shape and curvature scales. *Image and vision computing*, 10(8):557–564, 1992.

[60] Vijay Konda and John Tsitsiklis. Actor-critic algorithms. *Advances in neural information processing systems*, 12, 1999.

[61] Joseph B Kruskal. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, 29(1):1–27, 1964.

[62] Joseph B Kruskal. Nonmetric multidimensional scaling: a numerical method. *Psychometrika*, 29(2):115–129, 1964.

[63] Gottfried Wilhelm Leibniz. *Meditatio nova de natura anguli contactus et osculi: horumque usu in practica mathesi, ad figuras faciliores succedaneas difficilioribus substituendas (New meditation on the nature of the angles of tangency and osculation and their mathematical application in order to successfully replace complex figures with simpler ones).* Number 7. 1686.

[64] Qi Li and Guiping Hu. Multistage stochastic programming modeling for farmland irrigation management under uncertainty. *Plos one*, 15(6):e0233723, 2020.

[65] Stan Lipovetsky and Michael Conklin. Analysis of regression in game theory approach. *Applied Stochastic Models in Business and Industry*, 17(4):319–330, 2001.

[66] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation forest. In *2008 Eighth IEEE International Conference on Data Mining*, pages 413–422, 2008.

[67] Nils Lohndorf and Alexander Shapiro. Modeling time-dependent randomness in stochastic dual dynamic programming. *European Journal of Operational Research*, 273(2):650–661, 2019.

[68] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 4765–4774. Curran Associates, Inc., 2017.

[69] M. Maceira and C. Bezerra. Stochastic streamflow model for hydroelectric systems. In *Int. Conf. on Probabilistic Methods Applied to Power Systems (PMAPS)*, pages 637–645, 1997.

[70] M. Maceiral, D. Penna, A. Diniz, R. Pinto, A. Melo, C. Vasconcellos, and C. Cruz. Twenty years of application of stochastic dual dynamic programming in official and agent studies in brazil-main features and improvements on the NEWAVE model. In *Power Systems Computation Conf. (PSCC)*, pages 1–7, 2018.

[71] Spyros Makridakis. Accuracy measures: Theoretical and practical concerns. *International journal of forecasting*, 9(4):527–529, 1993.

[72] L. Martins and A. Azevedo amd S. Soares. Nonlinear medium-term hydro-thermal scheduling with transmission constraints. *IEEE Transactions on Power Systems*, 29(4):1623–1633, 2014.

[73] Diganta Misra. Mish: A self regularized non-monotonic activation function, 2020.

[74] Guido F Montufar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. On the number of linear regions of deep neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014.

[75] Ian D Moore, Paul E Gessler, GAE Nielsen, and GA Peterson. Soil attribute prediction using terrain analysis. *Soil science society of america journal*, 57(2):443–452, 1993.

[76] Saral Mukherjee and A.K. Chatterjee. The average shadow price for milps with integral resource availability and its relationship to the marginal unit shadow price. *European Journal of Operational Research*, 169(1):53–64, 2006.

[77] F. Nazare and A. Street. Solving multistage stochastic linear programming via regularized linear decision rules: An application to hydrothermal dispatch planning. *CoRR*, abs/2110.03146, 2021.

[78] nbro. How can the convolution operation be implemented as a matrix multiplication? Artificial Intelligence Stack Exchange, 2020. [Online:] https://ai.stackexchange.com/a/21874.

[79] Mathias Niepert, Pasquale Minervini, and Luca Franceschi. Implicit MLE: Backpropagating through discrete exponential family distributions. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021.

[80] Georg Simon Ohm. *Die Galvanische Kette: Mathematisch Bearbeitet (The Galvanic Circuit Investigated Mathematically)*. TH Riemann, 1827.

[81] W. Oliveira, C. Sagastizábal, D. Jardim Penna, M. Maceira, and J. Damázio. Optimal scenario tree reduction for stochastic streamflows in power generation planning problems. *Optim. Methods Softw.*, 25(6):917–936, 2010.

[82] Jose Javier Gonzalez Ortiz, John Guttag, and Adrian Dalca. Non-proportional parametrizations for stable hypernetwork learning. *arXiv preprint arXiv:2304.07645*, 2023.

[83] Sergei Ovchinnikov. Max-min representation of piecewise linear functions, 2000.

[84] M. Panteli and P Mancarella. Influence of extreme weather and climate change on the resilience of power systems: Impacts and possible mitigation strategies. *Electric Power Systems Research*, 127:259–270, 2015.

[85] M. Pereira and L. Pinto. Stochastic optimization of a multireservoir hydroelectric system: A decomposition approach. *Water Resources Research*, pages 779–792, 1985.

[86] M. Pereira and L. Pinto. Multi-stage stochastic optimization applied to energy planning. *Mathematical Programming*, 52(2):359–375, 1991.

[87] M.V.F. Pereira. Optimal stochastic operations scheduling of large hydroelectric systems. *International Journal of Electrical Power  Energy Systems*, 11(3):161–169, 1989.

[88] A.B. Philpott and V.L. de Matos. Dynamic sampling algorithms for multi-stage stochastic programs with risk aversion. *European Journal of Operational Research*, 218(2):470–483, 2012.

[89] Andrew B Philpott, Faisal Wahid, and J Frédéric Bonnans. Midas: A mixed integer dynamic approximation scheme. *Mathematical Programming*, 181(1):19–50, 2020.

[90] Marin Vlastelica Pogancic, Anselm Paulus, Vit Musil, Georg Martius, and Michal Rolinek. Differentiation of blackbox combinatorial solvers. In *International Conference on Learning Representations*, 2020.

[91] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144, 2016.

[92] Bernhard Riemann. *Grundlagen für Eine Allgemeine Theorie der Functionen Einer Veränderlichen Complexen Grösse (Foundations for a general theory of functions of a complex variable)*. Huth, 1851.

[93] Andy Roberts. Curvature attributes and their application to 3d interpreted horizons. *First break*, 19(2):85–100, 2001.

[94] Sam T Roweis and Lawrence K Saul. Nonlinear dimensionality reduction by locally linear embedding. *science*, 290(5500):2323–2326, 2000.

[95] Cynthia Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, 1(5):206–215, 2019.

[96] Susan M Schoenung, James M Eyer, Joseph J Iannucci, and Susan A Horgan. Energy storage for a competitive power market. *Annual review of energy and the environment*, 21(1):347–370, 1996.

[97] Sven Schönherr. *Quadratic programming in geometric optimization: Theory, implementation, and applications*. PhD thesis, ETH Zurich, 2002.

[98] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[99] Hans Ivar Skjelbred, Jiehong Kong, and Olav Bjarte Fosso. Dynamic incorporation of nonlinearity into milp formulation for short-term hydro scheduling. *International Journal of Electrical Power  Energy Systems*, 116:105530, 2020.

[100] M. Soares, A. Street, and V. Ao. On the solution variability reduction of stochastic dual dynamic programming applied to energy planning. *European Journal of Operational Research*, 258(3):743–760, 2017.

[101] S. Soares, C. Lyra, and H. Tavares. Optimal generation scheduling of hydrothermal power systems. *IEEE Transactions on Power Apparatus and Systems*, 6(3):1107–1118, 1980.

[102] A. Soroudi and T. Amraee. Decision making under uncertainty in energy systems: State of the art. *Renewable and Sustainable Energy Reviews*, 28:376–384, 2013.

[103] SA Stewart and R Podolski. Curvature analysis of gridded geological surfaces. *Geological Society, London, Special Publications*, 127(1):133–147, 1998.

[104] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.

[105] Joshua B Tenenbaum, Vin de Silva, and John C Langford. A global geometric framework for nonlinear dimensionality reduction. *science*, 290(5500):2319–2323, 2000.

[106] Warren S Torgerson. Theory and methods of scaling. 1958.

[107] Georges Voronoi. Nouvelles applications des paramètres continus à la théorie des formes quadratiques. deuxième mémoire. recherches sur les parallélloèdres primitifs. (new applications of continuous parameters to the theory of quadratic forms.). *Journal für die reine und angewandte Mathematik (Crelles Journal)*, 1908(134):198–287, 1908.

[108] Dong Wang, Yikai Chen, Changqing Shen, Jingjing Zhong, Zhike Peng, and Chuan Li. Fully interpretable neural network for locating resonance frequency bands for machine condition monitoring. *Mechanical Systems and Signal Processing*, 168:108673, 2022.

[109] Yuhui Wang, Hao He, and Xiaoyang Tan. Truly proximal policy optimization. In *Uncertainty in Artificial Intelligence*, pages 113–122. PMLR, 2020.

[110] Florian Wetschoreck, Tobias Krabel, and Surya Krishnamurthy. 8080labs/ppscore: zenodo release, October 2020.

[111] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning*, pages 2048–2057. PMLR, 2015.

[112] Ryo Yonetani, Tatsunori Taniai, Mohammadamin Barekatain, Mai Nishimura, and Asako Kanezaki. Path planning using neural a* search. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 12029–12039. PMLR, 18–24 Jul 2021.

[113] Behzad Zahiri, S Ali Torabi, Mehrdad Mohammadi, and Mohsen Aghabegloo. A multi-stage stochastic programming approach for blood supply chain planning. *Computers & Industrial Engineering*, 122:1–14, 2018.

[114] Quanshi Zhang, Ying Nian Wu, and Song-Chun Zhu. Interpretable convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8827–8836, 2018.

[115] Jikai Zou, Shabbir Ahmed, and Xu Andy Sun. Stochastic dual dynamic integer programming. *Mathematical Programming*, 175:461–502, 2019.