Leveraging convergence behavior to balance conflicting tasks in multi-task learning

Angelica Tiemi Mizuno Nakamura

Tese de Doutorado do Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional (PPG-CCMC)

Angelica Tiemi Mizuno Nakamura

Leveraging convergence behavior to balance conflicting tasks in multi-task learning

Thesis submitted to the Instituto de Ciências Matemáticas e de Computação – ICMC-USP – in accordance with the requirements of the Computer and Mathematical Sciences Graduate Program, for the degree of Doctor in Science. *FINAL VERSION*

Concentration Area: Computer Science and Computational Mathematics

Advisor: Prof. Dr. Denis Fernando Wolf

USP – São Carlos
January 2023

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi
e Seção Técnica de Informática, ICMC/USP,
com os dados inseridos pelo(a) autor(a)

```
        Nakamura, Angelica Tiemi Mizuno
N163l      Leveraging convergence behavior to balance
        conflicting tasks in multi-task learning / Angelica
        Tiemi Mizuno Nakamura; orientador Denis Fernando
        Wolf. -- São Carlos, 2023.
           87 p.


           Tese (Doutorado - Programa de Pós-Graduação em
        Ciências de Computação e Matemática Computacional) --
         Instituto de Ciências Matemáticas e de Computação,
        Universidade de São Paulo, 2023.


           1. Aprendizado de múltiplas tarefas. 2. Redes
        Neurais. 3. Otimização multi-objetivo. I. Wolf,
        Denis Fernando, orient. II. Título.
```

Bibliotecários responsáveis pela estrutura de catalogação da publicação de acordo com a AACR2:
Gláucia Maria Saia Cristianini - CRB - 8/4938
Juliana de Souza Moraes - CRB - 8/6176

Angelica Tiemi Mizuno Nakamura

Aproveitando o comportamento de convergência para equilibrar tarefas conflitantes no aprendizado de múltiplas tarefas

Tese apresentada ao Instituto de Ciências Matemáticas e de Computação – ICMC-USP, como parte dos requisitos para obtenção do título de Doutora em Ciências – Ciências de Computação e Matemática Computacional. *VERSÃO REVISADA*

Área de Concentração: Ciências de Computação e Matemática Computacional

Orientadora: Prof. Dr. Denis Fernando Wolf

USP – São Carlos
Janeiro de 2023

Dedicated to my family. Especially to my parents, Hiroshi and Aquemi, to my brother Diego, to my grandparents Yoshimi and Shinkichi (in memorian), and Tocie (in memorian) and Toshio (in memorian).

ACKNOWLEDGEMENTS

A Ph.D. is a long journey and I am grateful to all the people who crossed my path and who somehow contributed to this process.

First of all, I would like to express my gratitude to professors Denis Wolf and Valdir Grassi Junior. Thank you very much for all the advice and patience that guided me to grow and achieve my goals. I could not have undertaken this journey without you.

Takeshi Horita, thank you for always being by my side. To be my personal and intellectual partner, and for supporting me whenever I needed it. You always make my endeavors and challenges lighter to go through.

Members of Mobile Robotics Lab, I am so grateful to have worked in a group with amazing people. Thank you so much for all the discussions, friendship, and birthday parties.

I am also grateful for the support given by the São Paulo Research Foundation - FAPESP [grant number 2019/03366-2]. This study was financed in part by the Coordination for the Improvement of Higher Education Personnel - Brazil (CAPES) - Finance Code 001.

Finally, my immeasurable gratitude to my family. Especially to my parents and brother, for giving me all the base and never measuring efforts so that I could reach my goals, for being my inspiration, and for always believing in me.

“If they give you ruled paper,
write the other way.”
(Ramón Jiménez)

RESUMO

NAKAMURA, A. T. M. **Aproveitando o comportamento de convergência para equilibrar tarefas conflitantes no aprendizado de múltiplas tarefas**. 2023. 87 p. Tese (Doutorado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2023.

O aprendizado de múltiplas tarefas é um paradigma de aprendizagem que utiliza tarefas correlacionadas para melhorar a generalização. Uma maneira comum de aprender várias tarefas é por meio da abordagem com parâmetros compartilhados, na qual uma única arquitetura é usada para compartilhar o mesmo subconjunto de parâmetros, criando um viés indutivo entre eles durante o processo de treinamento. Devido à sua simplicidade, pontencial em melhorar a generalização e reduzir o custo computacional, o aprendizado de múltiplas tarefas ganhou a atenção das comunidades científica e indústria. Na literatura, o aprendizado simultâneo de múltiplas tarefas é normalmente realizado por uma combinação linear de funções de perda. No entanto, os gradientes das tarefas frequentemente conflitam entre si durante a otimização das funções de perdas. E, combinar os gradientes de todas as tarefas para que todas convirjam para sua solução ótima ao longo do processo de treinamento não é trivial.

Para resolver este problema, é utilizado a ideia de otimização multi-objetivo para propor um método que leva em conta o comportamento temporal dos gradientes para criar um viés dinâmico que ajusta a importância de cada tarefa durante a retropropagação. Dessa forma, o método dá mais atenção para as tarefas que estão divergindo ou não sendo beneficiadas nas últimas iterações, garantindo que o aprendizado simultâneo alcance a maximização do desempenho de todas as tarefas. Para validar o método proposto, foram realizados análise de sensibilidade e diversos experimentos no conjunto de dados público de classificação de dígitos, e no problema de compreensão de cena no conjunto de dados do CityScapes. Por meio dos experimentos realizados, o método proposto mostrou superar o desempenho dos métodos estado da arte na aprendizagem de tarefas conflitantes, garantindo que todas as tarefas alcancem bons desempenhos de generalização ao mesmo tempo em que acelera a convergência das curvas de aprendizado.

Palavras-chave: Aprendizado de múltiplas tarefas, Redes neurais, Otimização multi-objetivo.

ABSTRACT

NAKAMURA, A. T. M. **Leveraging convergence behavior to balance conflicting tasks in multi-task learning**. 2023. 87 p. Tese (Doutorado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2023.

Multi-Task Learning is a learning paradigm that uses correlated tasks to improve performance generalization. A common way to learn multiple tasks is through the hard parameter sharing approach, in which a single architecture is used to share the same subset of parameters, creating an inductive bias between them during the training process. Due to its simplicity, potential to improve generalization, and reduce computational cost, it has gained the attention of the scientific and industrial communities. In the literature, the simultaneous learning of multiple tasks is usually performed by a linear combination of loss functions. Nonetheless, tasks' gradients often conflict with each other during losses' optimization, and it is not trivial to combine them so that all tasks converge toward their optimal solution throughout the training process. To address this problem, the idea of multi-objective optimization was adopted to propose a method that takes into account the temporal behavior of the gradients to create a dynamic bias that adjusts the importance of each task during backpropagation. The result of this method is to give more attention to tasks that are diverging or not being benefited during the last iterations, ensuring that the simultaneous learning is heading to the performance maximization of all tasks. To evaluate the performance of the proposed method in learning conflicting tasks, sensitivity analysis and a series of experiments were performed on a public handwritten digit classification dataset, and on the scene understanding problem in the CityScapes Dataset. Through the performed experiments, the proposed method outperformed state-of-the-art methods in learning conflicting tasks. Unlike the adopted baselines, the proposed method ensures that all tasks reach good generalization performances at the same time it speeds up the learning curves.

Keywords: Multi-task learning, Neural networks, Multi-objective optimization.

LIST OF FIGURES

LIST OF ALGORITHMS

LIST OF TABLES

LIST OF SYMBOLS

k_m — Weighting coefficient of task m

M — Number of objective functions

$||x||$ — L2-norm of x

$\nabla f_m(\cdot)$ — Gradient vector of the m^{th} objective function

$\bar{\nabla}_{sh,m}$ — Normalized gradients of task m w.r.t. the shared parameters

u_m — Tension vector of a task m

c_m — Tension factor of a task m

t — Current time-step

T — Period o iterations

$\zeta_m(t)$ — Average of gradients' magnitude at a time t

δ_m — Relative variation of a task m

$\vec{v}$ — Common descent direction

$\vec{v}_n$ — Adapted common descent direction with the use of tensioners

CONTENTS

CHAPTER

1

INTRODUCTION

Machine learning can be used on several types of applications, and many of them may require several abilities. For example, self-driving cars must be able to detect obstacles for collision avoidance, detect navigable areas and horizontal lane markings for lane keeping, detect and recognize traffic signs to respect traffic rules, and so on. Each of these abilities can be interpreted as a task to be performed simultaneously. One common way of resolving multiple tasks is by implementing one solution for each task and running all of them in parallel. However, this approach can be computationally very costly. Alternatively, there is a machine learning concept that allows simultaneous learning of multiple tasks, i.e., Multi-Task Learning (MTL). This approach can potentially reduce inference latency and computational storage when using a single architecture by sharing common knowledge among related tasks. Moreover, MTL can also be used to explore the idea of creating an inductive bias between tasks (CARUANA, 1997). Due to its potential to improve generalization performance, it has gained attention in several areas of scientific and industrial communities, such as computer vision (SENER; KOLTUN, 2018; CIPOLLA; GAL; KENDALL, 2018; CHEN *et al.*, 2018; NAKAMURA; GRASSI; WOLF, 2021) and natural language processing (MAO *et al.*, 2020; LIU; QIU; XUANJING, 2016; XIAO; ZHANG; CHEN, 2018).

In deep neural networks, MTL can be categorized according to how parameters are shared between tasks, i.e., soft and hard parameter sharing approaches. In soft parameter sharing, each task has its own model with its own parameters, and the inductive bias occurs only by a regularization function between models. On the other hand, the hard parameter sharing approach creates inductive bias through multiple tasks by sharing the same subset of parameters. This work focuses on the second approach.

The main advantage of hard parameter sharing is related to the fact that multiple tasks are learned using a single network. It allows the implementation of simpler solutions to reduce computational cost and inference latency; at the same time, it potentially improves generalization by implicit bias induction. In this approach, the simultaneous learning of multiple tasks is usually performed by a linear combination of loss functions (EIGEN; FERGUS, 2015; TEICHMANN *et al.*, 2018; CHENNUPATI *et al.*, 2019). Therefore, the tasks will share and compete for the same resource (e.g., neurons or convolutional kernels), making it necessary to handle conflicting tasks. In this work, we define two tasks as conflicting when the gradients of their losses can have different directions during training, i.e., the angle between gradients is greater than 0 degree. By this definition, we can say that the conflict level increases as the angle between gradients gets closer to 180 degrees. Thus, when the descent direction of a task is not representative for the other, optimizing the network parameters in relation to a bad direction can negatively impact the performance of some tasks. Therefore, it is necessary to find a trade-off between tasks, which is beyond what a linear combination can achieve.

As an alternative, Sener and Koltun (2018) proposed to model MTL as a multi-objective optimization problem to find a solution that is not dominated by any other. That is, to find a trade-off in which it is not possible to improve one task without degrading another. The problem is that, in multi-objective optimization, there is no single solution that is optimal for all tasks, but several solutions with different trade-offs between them (MIETTINEN, 1998; EHRGOTT, 2005). Furthermore, considering problems of different natures, variations in the task's gradient scales can influence the definition of the solution. Thus, when a solution is said to be optimal, it does not mean that all tasks will be performing well, but only that the solution has reached a point where no further improvement can be made to all tasks simultaneously. This idea will be better explained in Chapters 2 and 4.

Based on the above considerations, in this thesis, we propose a method that guarantees better performances for all tasks without giving up the optimality condition of multi-objective optimization. The difference between the existing methods and the proposed one is how gradients are combined.

Existing methods only consider the gradients computed in the current iteration to estimate an appropriate combination of them in order to obtain a common descent direction between tasks. However, since these methods get only a snapshot of the current gradients, they need to create a static bias when combining the gradients, such as getting the direction with the minimum norm (SENER; KOLTUN, 2018) or the central direction between gradients (KATRUTSA *et al.*, 2020). Because of this, they cannot ensure that all tasks are converging to their optimal performances, although they are meeting the optimality conditions. On the other hand, we hypothesize that, if the temporal factor of the gradients were considered to compute their combination on the current iteration, it would be possible to analyze whether the performances of all tasks are converging or not and create a dynamic bias.

1.1 Contributions

Based on the above considerations, we propose a method that creates the idea of tensioners that strengthen when their respective tasks lose performance and weaken when they increase performance. This is done at the same time the optimality conditions are met. By doing so, the combination of gradients is modulated to give preference to those that are diverging from their optimal performance, ensuring that all tasks will converge together.

Therefore, the main contributions proposed in this Ph.D. thesis are as follows:

- An approach independent of a specific application to combine multiple gradient vectors and finding a common feasible descent direction, aiming to maximize the performance of all tasks.
- A method to create a dynamic learning bias by leveraging the convergence behavior of tasks.

To confirm the performance of the proposed method in learning conflicting tasks, we performed sensitivity analysis and a series of experiments on a public handwritten digit classification dataset, and on the scene understanding problem in the CityScapes Dataset.

1.2 Thesis Outline

This thesis is structured as follows. Chapter 2 provides a theoretical background of multi-task learning and multi-objective optimization. Chapter 3 covers the studies in the area of multi-task learning. Chapter 4 presents our proposed method to combine gradients of multiple tasks by leveraging their convergence behavior over the past iterations. Chapter 5 describes metrics, data, and experimental setup. Also, it presents sensitivity analysis and compares quantitative and qualitatively our proposed method with other weighting methods, followed by a discussion of the results. Finally, Chapter 6 concludes with an overall discussion and possible improvements. Published journals and papers are shown in the Appendix section.

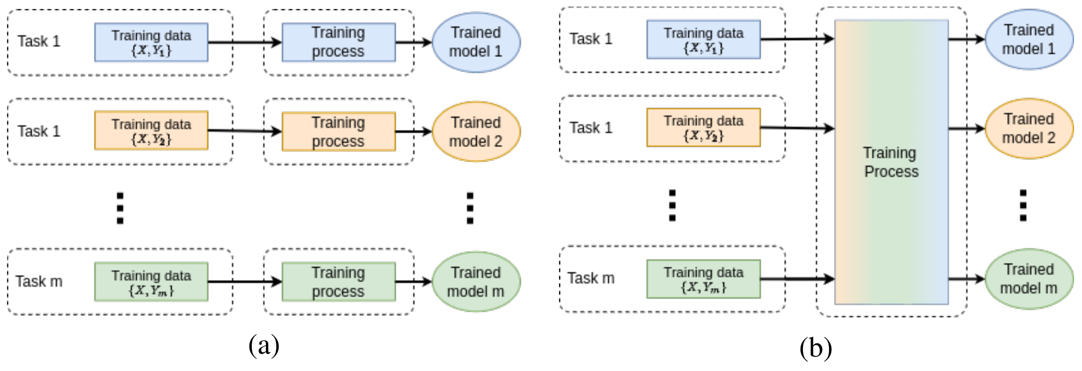
CHAPTER 2

THEORETICAL BACKGROUND

In machine learning, when trying to solve a complex problem, one could break it into sub-problems to solve each of them separately (Figure 1a). For example, a self-driven car must be able to understand the environment in which it is navigating. For this, the perception system can be broken into separate solutions, like navigable area detection, obstacles detection, traffic signs detection, and recognition, among others (ROSERO *et al.*, 2020). However, this approach can be computationally very costly and does not explore the source of information contained in training signals from other tasks. Alternatively, in this thesis, we propose to explore a machine learning concept that allows simultaneous learning of multiple tasks by sharing common knowledge among related tasks during the training process, i.e., Multi-Task Learning (MTL) (Figure 1b).

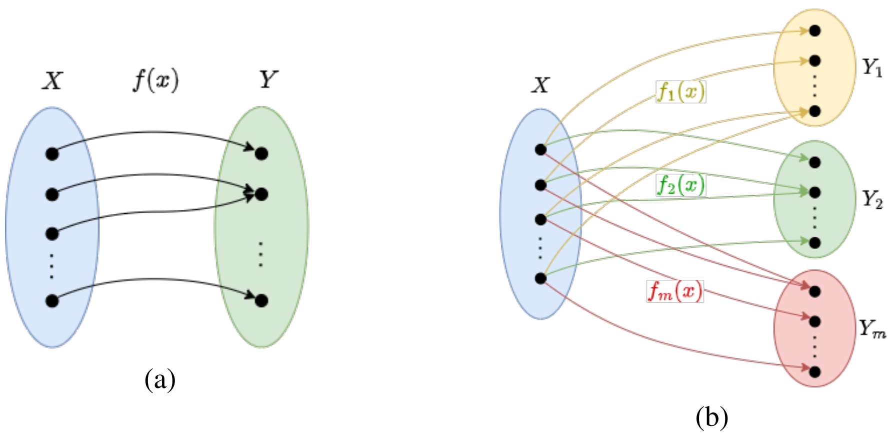
To provide a brief theoretical background, this chapter is divided into two subsections. First, Section 2.1 presents some definitions and formulation of multi-task learning architectures with a hard parameter sharing approach. In this approach, the simultaneous learning of multiple tasks is given by the optimization of the shared parameters in function of the combination of

Figure 1 – Single task learning vs. Multi-task learning framework. (a) The models are learned individually for each task, not sharing information between tasks. (b) The relatedness among correlated tasks is shared during training by joint learning.



Source: Elaborated by the author.

Figure 2 – (a) Multi-output and multi-label learning approximate a function, $f : X \to Y$, that maps data points from the input domain to a unique output image domain, where each point is represented by M dimensions given by M outputs or labels; (b) Multi-task learning approximates M different functions, $f_m : X \to Y_m$, that maps data points from one input domain to M output image domains, where each of them corresponds to a distinct task.



Source: Elaborated by the author.

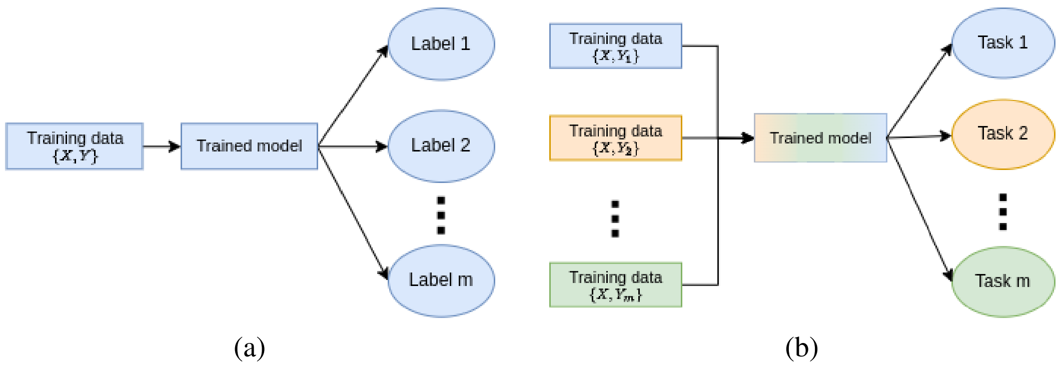
gradient vectors of each task. Therefore, to prevent the problem of one task dominating the other during training, we present in sequence the concept of multi-objective optimization to combine multiple gradients of different tasks (Section 2.2).

2.1 Multi-task Learning

Introduced by Caruana (1993), MTL is a machine learning paradigm in which multiple related tasks are learned simultaneously, such that each task can contribute to a shared knowledge that enhances the generalization performance of every other task at hand. This idea of learning multiple tasks can be related to other paradigms in ML, such as multi-output regression, multi-label classification, and transfer learning. One could say that MTL applied to neural networks is simply the fact of creating an architecture with multiple outputs to solve regression or classification tasks, which has already been extensively explored. If all the possible labels were treated as a task, multi-output and multi-label learning can be considered as a special case of MTL. However, the output variables in this kind of solution usually share the same features or data during training (LECUN *et al.*, 1989; CARUANA, 1997; ZHANG; YANG, 2021) (Figure 2).

On the other hand, in MTL with a hard parameter sharing approach, different tasks possess different data and share only the initial layers that describe lower-level features, being necessary to keep some high-level feature descriptors for each task (Figure 3). We can consider as an example the problem of simultaneous learning of instance segmentation and depth estimation to explore the relatedness between them and improve the overall performance (NAKAMURA; GRASSI; WOLF, 2021). In this case, the discontinuity of distance values among objects provides

Figure 3 – Architecture differences between (a) Multi-output and multi-label learning, and (b) Multi-task learning.



Source: Elaborated by the author.

the silhouette of the instances, while the object instances help to make depth predictions smooth and continuous. Although both tasks can be modeled as regression problems, they have different training data and need to extract different features to get simultaneous predictions from the respective tasks.

In general, the idea of MTL is that related tasks can create an inductive bias, and based on that bias, a certain task will prefer some hypothesis over the n possible hypotheses (CARUANA, 1993). Because MTL leverages knowledge from other tasks to improve the learning process, it can also be confused with transfer learning. The difference between them lies in how the knowledge is used. While in transfer learning, the knowledge of an already trained model is used directly in the new neural network model, in MTL, the knowledge is consolidated during the learning process through the simultaneous backpropagation of different tasks gradients (Figure 4).

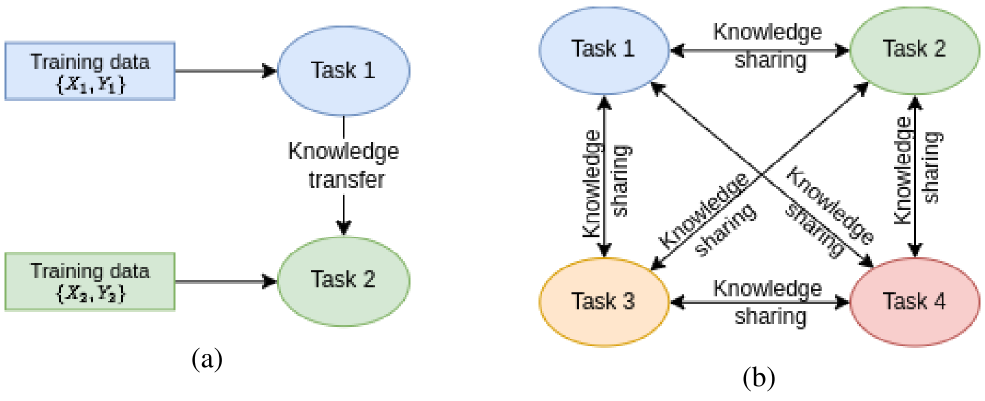
Therefore, considering a supervised MTL problem in a hard parameter sharing approach, a task m is described by a set of data $(\mathbf{x}_i, \mathbf{y}_{m,i})$, where $\mathbf{x}_i \in X$ is the i^{th} training instance and $\mathbf{y}_{m,i} \in \{Y_m\}_{m \in M}$ is the corresponding label for task m. To perform the simultaneous learning of multiple tasks considering the problem over a space X and a set of labels $\{Y_m\}_{m\in[M]}$, the decision function of each task m can be defined as $f_m(x; \mathbf{w}_{sh}; \mathbf{w}_m) : X \to Y_m$. Where the $\mathbf{w}_{sh}$ parameters are shared among tasks and $\mathbf{w}_m$ are task-specific parameters.

Hence, given a set of objective functions $\mathscr{L}(f_m(x; \mathbf{w}_{sh}, \mathbf{w}_m))$ that defines the empirical risk of the task m, the MTL problem is commonly formulated as the minimization of the linear combination of objective functions:

$$\min_{\substack{\mathbf{w}_{sh},\\ \mathbf{w}_1,\dots,\mathbf{w}_M}} \frac{1}{N} \sum_{i=1}^{N} \sum_{m=1}^{M} k_m \mathscr{L}(f_m(x_i; \mathbf{w}_{sh}, \mathbf{w}_m), y_{i,m}), \tag{2.1}$$

where N is the number of data and $k_m > 0$ are the predefined or dynamically calculated weighting coefficients associated with each objective function. Predefining those weighting coefficients

Figure 4 – Difference between transfer learning and multi-task learning. (a) The information flows in only one direction, which is from the source task to the destination task. (b) The information can flow in any direction between all tasks.



Source: Elaborated by the author.

can be cumbersome and prone to errors. On the other hand, one can model this problem as a Multi-objective Optimization (MOO) to adjust those weights iteratively. To understand how it could be done, the next section introduces the concept of MOO.

2.2 Multi-objective Optimization

In machine learning, we commonly work by optimizing an objective function trying to minimize its error or maximize its gain. Multi-objective optimization (MOO) extends this concept of single-objective optimization, in which more than one objective function must be simultaneously optimized. This section is based on the books: Miettinen (1998), Hwang and Masud (1979) and Ehrgott (2005).

In the literature, the problem of MOO can be defined as:

$$\begin{aligned} \text{Minimize} \quad & \mathbf{z} = (f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_M(\mathbf{x})), \\ \text{subject to} \quad & \mathbf{x} \in S, \end{aligned} \tag{2.2}$$

where $M \geq 2$ is the number of objective functions, $\mathbf{x}$ is the decision vector, and S is the feasible set of decision vectors defined by constraint functions, which is a subset of the $\mathbb{R}^n$ decisions space.

Furthermore, each point in the feasible set S has a corresponding point in the objective space $\mathbb{R}^M$, called the set of feasible objectives Z. The elements of Z are the objective vectors, $\mathbf{z} = (z_1, z_2, \ldots, z_M)$, where $z_m = f_m(\mathbf{x})$ for all $m = 1, \ldots, M$, are the objective values.

In problem 2.2, we want to minimize all objective functions simultaneously. Then, if there was no conflict between functions, the solution to this problem would be when all functions reach their optimum, not being necessary to find a balance to optimize the functions. However, this work focuses on problems in which there is a conflict between functions, that is, minimizing one function may imply increasing the other.

For example, consider the classic problem in which we want to buy a car, and we want the cheapest car that consumes less fuel. Figure 5a shows the car models in the decision space belonging to the feasible set and its corresponding points in objective space, i.e., the models that comply with defined restrictions. In this case, the restrictions are the size and comfort of the car.

When the MOO problem has conflicting functions, the solution to this problem is not always to simultaneously reach the optimum for all functions, since the decrease of one objective function may imply the increase of another. For example, choosing model 4 instead of model 3 to reduce the purchase price implies increasing fuel consumption (Figure 5b).

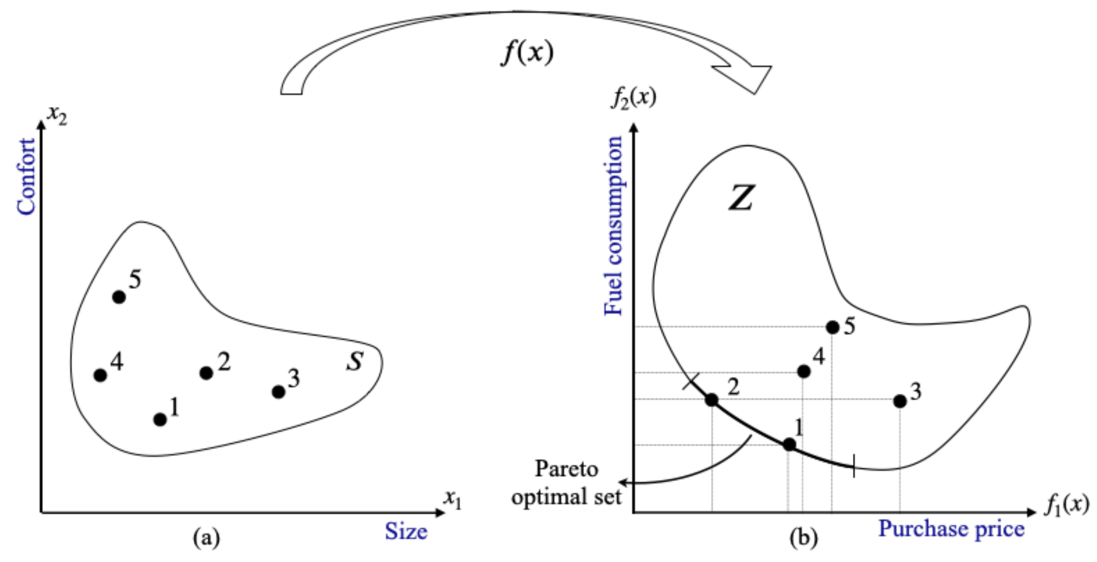
This is the general idea of MOO, where there will not be a single solution that will be optimal in relation to all objectives, but a set of efficient solutions (Pareto-optimal) that will not improve one objective function without degrading at least one of the other objectives.

Definition 1 (Pareto-optimality)**.** A decision vector $\mathbf{x}^* \in S$ is Pareto optimal if there does not exist another decision vector $\mathbf{x} \in S$ such that:

1. $f_m(\mathbf{x}) \leq f_m(\mathbf{x}^*), \forall m \in \{1, \ldots, M\};$

2. $f_j(\mathbf{x}) < f_j(\mathbf{x}^*),$ for some $j \in \{1, \ldots, M\},$

Figure 5 – Mapping the decision space in the objective space. Each point in the decision space (left) represents a solution and corresponds to a certain point in the objective space (right).



Source: Adapted from Miettinen (1998).

Therefore, a decision vector **x*** is Pareto-optimal if there is no other solution that simultaneously improves all m objective functions, and also, if there is no other function that improves at least one objective function (MIETTINEN, 1998; EHRGOTT, 2005; HWANG; MASUD, 1979). In the example of Figure 5, the two non-dominated solutions are models 1 and 2, since there is no other solution that is better in relation to the two models.

There are several MOO methods (EHRGOTT, 2005) with different conditions on the function $f : \mathbb{R}^n \to \mathbb{R}^M$ and feasible set $S \subseteq \mathbb{R}^n$ that guarantee finding a Pareto-optimal solution. However, this thesis will focus on the steepest descent methods (FLIEGE; SVAITER, 2000), which can be considered as an extension of scalar optimization methods. Hence, considering a minimization problem, the objective values will decrease at each iteration in the partial order induced by the cone formed by adjacent gradients (FUKUDA; DRUMMOND, 2014).

Translating it to the MTL scenario, we can assume the loss functions of all tasks as the objective functions to be minimized and the shared parameters of the network as the decision vector. To allow finding a Pareto-optimal solution, we need to find a descent direction that fits inside the cone formed by the negative gradients of the loss functions of all tasks. This condition could be enough if the combination of all tasks was convex (i.e. if all tasks were not conflicting), but it is not always true. To guarantee the simultaneous optimization of the objective functions, we also need to analyze the feasibility of the descent direction, which is explained in the next section.

2.2.1 Feasible Descent Direction

According to Fliege and Svaiter (2000), a direction $\vec{v}$ can simultaneously improve all objective functions if:

$$-\nabla f_m(\mathbf{x}) \cdot \vec{v} \geq 0, \forall m \in 1, \ldots, M, \tag{2.3}$$

where $\nabla f_m(\mathbf{x})$ represents the gradient vector of the m^{th} objective function. Considering that the dot product of any two vectors $\vec{p}, \vec{q} \in \mathbb{R}^n$ can also be written as $\vec{p} \cdot \vec{q} = ||\vec{p}|| ||\vec{q}|| \cos(\theta)$, where θ is the angle formed between the vectors, it means that the angle between the negative vector of $\nabla f_m(\mathbf{x})$ and $\vec{v}$ must be smaller than or equal to 90 degrees, since $-\nabla f_m(\mathbf{x})) \cdot \vec{v} \geq 0$.

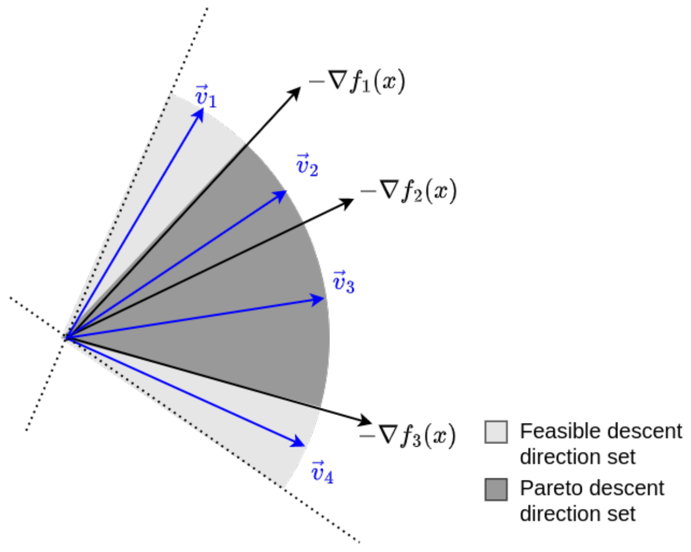
For better understanding, assume the example in Figure 6 where there are three black vectors representing the negative gradients of the tasks, $\nabla f_m(\mathbf{x})$. In this scenario, there will be a range of feasible descent directions that attends the condition of the Equation 2.3. This range is represented by the light and dark gray regions. The blue vectors, $\vec{v}_i$, represent some samples of feasible directions, which are normalized to remove the influence of their scales.

The definition of feasible descent direction guarantees that the direction will contribute at some level to the convergence of all tasks, but does not guarantee that it will be optimal. For this, the feasible descent direction should also attend to the Pareto-optimality conditions, i.e., it must be within a range of directions that defines non-dominated solutions. In other words, the directions cannot improve one solution without degrading any other. This set of solutions is represented by the descent directions within the dark gray region. For example, the descent direction $\vec{v}_2$ and $\vec{v}_3$ cannot improve an objective function without degrading the improvement of another:

$$\begin{aligned}\vec{v}_2 \cdot -\bar{\nabla} f_1(\mathbf{x}) > \vec{v}_3 \cdot -\bar{\nabla} f_1(\mathbf{x}),\\ \vec{v}_2 \cdot -\bar{\nabla} f_2(\mathbf{x}) > \vec{v}_3 \cdot -\bar{\nabla} f_2(\mathbf{x}),\\ \vec{v}_2 \cdot -\bar{\nabla} f_3(\mathbf{x}) < \vec{v}_3 \cdot -\bar{\nabla} f_3(\mathbf{x}),\end{aligned}$$

being necessary to find a trade-off between the objectives. These solutions are known as Pareto optimal solutions (the dark gray region in Figure 6).

Figure 6 – Feasible set and Pareto descent directions considering the negative gradient vectors of $\nabla f_1(\mathbf{x})$, $\nabla f_2(\mathbf{x})$ and $\nabla f_3(\mathbf{x})$. The blue vectors are examples of feasible descent directions given the set of negative gradients of the tasks. Best visualized with color.



Source: Adapted from Harada, Sakuma and Kobayashi (2006).

2.2.2 Pareto-optimal Descent Directions

A Pareto-optimal solution can be expressed as a convex combination of the negative gradients (HARADA; SAKUMA; KOBAYASHI, 2006; MIETTINEN, 1998). As given in Definition 1, a decision vector is said to be Pareto-optimal if there is no other solution that simultaneously improves all the objective functions, and also, there is no other function that improves at least one objective function. However, in most cases, it is not trivial to find those optimal solutions. This is because optimization problems in machine learning are usually highly nonlinear and nonconvex, and solving the global optimality becomes intractable (BOTTOU; CURTIS; NOCEDAL, 2018).

In steepest gradient methods, minimizing nonconvex objectives is more challenging because it can lead to many critical points (or stationary points). However, as shown by Bottou, Curtis and Nocedal (2018), when assuming a stochastic gradient or a Newton-like direction one can provide meaningful convergence guarantees. That said, the only thing we need is to define the feasible descent direction that leads to a Pareto-optimal solution in the context of MOO. For this, Désidéri (2012) introduced the notion of Pareto-stationarity as a necessary condition for Pareto-optimality.

Definition 2 (Pareto-stationarity)**.** Given M differentiable objective functions $f_m(\mathbf{x})$, $m = 1, \cdots, M$, $\mathbf{x} \in \mathbb{R}^d$, a point $\mathbf{x}^*$ is considered a Pareto-stationary point, if and only if there exists a convex combination of the gradients that is equal to zero, i.e.:

$$\textstyle\sum_{i=1}^{M} k_m \nabla f_m(\mathbf{x}) = 0, \quad \text{where } \sum_{i=1}^{M} k_m = 1, k_m \geq 0 \quad \forall m.$$

Therefore, if a given point $\mathbf{x}$ is not Pareto-stationary, the convex combination of its gradients will necessarily represent a descent direction for all objective functions (SCHÄFFLER; SCHULTZ; WEINZIERL, 2002; DÉSIDÉRI, 2012).

2.3 Final Considerations

This chapter presents the key theoretical backgrounds to understand the main contributions of this work. In MTL, each task is usually associated with a constant weighting coefficient, which was previously defined to allow simultaneous optimization of all functions. However, this configuration can negatively impact the model performance in situations where the tasks conflict with each other. To prevent this, we find a trade-off between multiple objective functions to properly optimize all tasks. Thus, we based on the concept of multi-objective optimization to find this balance between tasks. However, before jumping to the proposed method, the next chapter provides the necessary review of related works to support the state-of-art on MTL and the existing gap that we focused to develop this Ph.D. thesis.

CHAPTER

3

RELATED WORK

This chapter reviews the main works on MTL for neural networks available in the literature and highlights the insights and gaps that support the contribution of this Ph.D. thesis. The MTL can be analyzed in many ways and for a plethora of applications. Fortunately, there are several recently published surveys that can be referred (CRAWSHAW, 2020; ZHANG; YANG, 2021; CHEN; ZHANG; YANG, 2021; BAYOUDH *et al.*, 2021; REN *et al.*, 2021; VANDENHENDE *et al.*, 2022; SAMANT *et al.*, 2022). Therefore, the remaining of this chapter focuses on briefly reviewing the key points for this work, starting with a survey of architecture types that can be used for MTL followed by a review on optimization methods to learn multiple tasks simultaneously.

3.1 Multi-task Learning Architectures

As already discussed in Section 2.1, the core idea of MTL is that related tasks can create an inductive bias to help the convergence of other tasks. In deep learning, there are many ways of applying this idea, but all of them depend on how parameters are changed during the training process in function of the gradients computed over the loss function of each task. Essentially, the inductive bias is modeled according to neural network architectures, which historically have been divided into two main groups: the soft and the hard parameter sharing approaches.

In soft parameter sharing, each task has its own model with its own parameters. In this case, the bias induction is usually accomplished through functions that reduce the distance between the models' parameters (CHEN; ZHANG; YANG, 2021). For example, Guo *et al.* (2018) use the Euclidean distance to compare the difference between parameters, while Hai *et al.* (2016) adopted the covariance matrix. However, some other works propose methods to combine feature representations among tasks, like the Cross-stitch network that models shared representations as linear combinations (MISRA *et al.*, 2016). Another example is the SLUICE network (RUDER *et al.*, 2019), which controls what and how much feature subspaces should be shared by trainable sluice parameters. Considering that the set of parameters is responsible for modeling the feature extraction, it means that this group of methods encourages parallel networks of all tasks to learn at some level similar representation space. Because the networks are loosely coupled, this approach allows more flexibility for the tasks to learn common and task-specific features. Nonetheless, soft parameter sharing approaches have limited scalability because the amount of trainable parameters tends to increase linearly with the number of tasks (VANDENHENDE *et al.*, 2019).

Alternatively, the hard parameter sharing approaches allow a set of parameters to be literally shared among all tasks, reducing inference latency and storage costs (CARUANA, 1993). The first hard parameter sharing architecture and most common is the tree-like, in which the models of different tasks share the same feature extractor (or encoder) and have their own output layers (or decoders). This kind of architecture was initially proposed by Caruana (1993), which presented a multi-layer perceptron structure with a single input layer and multiple parallel outputs to learn different Boolean functions simultaneously. Empirically, the author showed that it is possible to increase the generalization performance when the tasks are trained together with the same feature layers. More recently, after the surge of Deep Learning, new architectures with tree-like structures have been created to attend different problem domains, like computer vision (BAYOUDH *et al.*, 2021) and natural language processing (CHEN; ZHANG; YANG, 2021). In any case, the central idea of sharing the base feature extractor among different related tasks to improve each single task keeps the same, and the only thing that changes is the input and output domains. For example, Zhang *et al.* (2014) proposed the Tasks-Constrained Deep Convolutional Network (TCDCN) to optimize facial landmark detection together with correlated tasks, arguing that representations learned from those related tasks facilitate the learning of the main task. Later, Kokkinos (2017) presented the UberNet, a universal fully convolutional network that allows low, mid, and high-level vision tasks learning to be applied to computer vision problems simultaneously. For this, the authors proposed to use separate skip layers from the shared encoder layers to each task decoder, in order to keep the task-specific memory and computation budget low at the same time that common features are learned. Concurrently, in natural language processing, Plaza-Del-Arco *et al.* (2021) proposed a tree-like architecture with a transformer-based model as the shared encoder for learning common features among hate speech detection, emotion detection, and polarity classification.

Besides the tree-like architecture, there are some other structure variations in hard parameter sharing that have some intersection with soft parameter sharing. For instance, the parallel feature fusion architecture allows parallel networks with task-specific layers that can be shared at some level to other tasks or can have some intermediate shared layers between tasks (CHEN; ZHANG; YANG, 2021). An example is the adversarial MTL proposed by Liu, Qiu and Huang (2017), in which each task has its own network to learn task-specific features, but also a shared adversarial network to learn shared features. To avoid redundant latent representations in relation to task-specific layers, the authors also proposed using orthogonality constraints to regularize the shared features. Another variation of the hard parameter sharing approach is the hierarchical architecture, where task-specific networks can explicitly combine features among them at different depths, including getting the output of one task to use as an extra input or additional control signal (CHEN; ZHANG; YANG, 2021). Following this idea, Vandenhende *et al.* (2019) proposed a method to share layers according to the analysis of affinity between tasks, which is computed through the Representation Similarity Analysis (DWIVEDI; ROIG, 2019).

Despite of the fact that many surveys have divided the MTL architectures into soft and hard parameters sharing, recent works took the advantage of the flexibility of artificial neural networks to design more different network configurations (VANDENHENDE *et al.*, 2022). For example, some works presented methods to automate procedures to design branched networks that best attend a certain set of tasks and a specific computational budget (REN *et al.*, 2021). Nonetheless, the networks generated by those methods are usually limited to task-specific models (VANDENHENDE *et al.*, 2019).

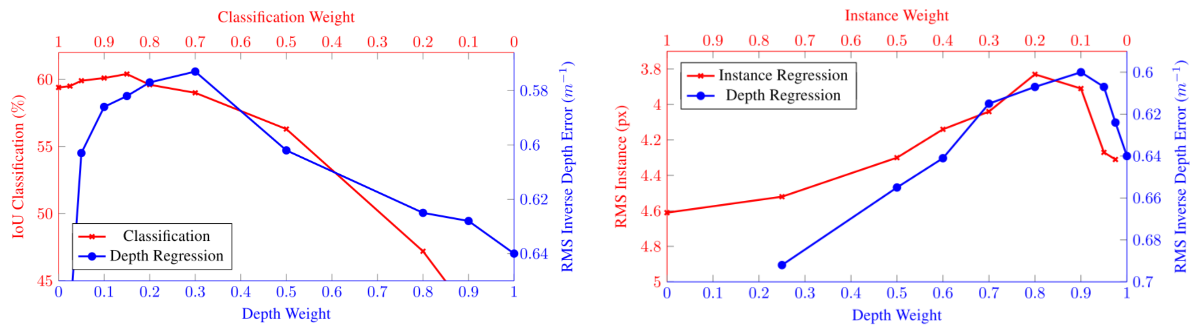
As in many scientific research topics, it is difficult to point out the best solution for designing a network architecture. According to Vandenhende *et al.* (2022), to make an appropriate choice of the network to be used, one needs to know the set of tasks that will be solved, the input and output domains, and every possible detail about their relationship. However, regardless of the architecture being used, the optimization strategy plays an essential role, especially in training the parameters that will be receiving the influence of biases from multiple tasks. The next section presents an overview of recent works on optimization methods for MTL.

3.2 Optimization for Multi-task Learning

The standard formulation of gradient descent-based methods for optimizing a neural network assumes only one objective function to be optimized. However, MTL presents multiple loss functions as objectives to be minimized simultaneously. In the literature, there are two different approaches for this problem. The first and the most common is the approach of combining multiple objective functions into a single objective function or combining the gradient of the loss functions. The second is to consider the MTL as a Multi-objective Optimization (MOO) problem.

The first approach is an approximation that allows using the standard formulation of a single optimization function. This is commonly done by a linear combination of the loss functions of all tasks (SERMANET *et al.*, 2013; MISRA *et al.*, 2016; KOKKINOS, 2017; TEICHMANN *et al.*, 2018; CHENNUPATI *et al.*, 2019; SANCHEZ *et al.*, 2019; LI *et al.*, 2021), or some variation with adaptive weights (CIPOLLA; GAL; KENDALL, 2018; CHEN *et al.*, 2018; LIU; JOHNS; DAVISON, 2019; LI *et al.*, 2016; GUO *et al.*, 2018). Although this approach is simple and has shown promising results, there are two inherent problems, as Gunantara (2018) pointed out. First, due to the fact that task-specific loss functions can have different magnitudes and behaviors, it becomes difficult to choose an appropriate set of weights for the linear combination. Secondly, because the combined loss function is used to update all the network parameters including those that are not shared among tasks, it would become a problem if the set of tasks is not convex. Moreover, the combination of losses may lead to information loss once it maps a set of loss values in $\mathbb{R}^n$ to a single value in $\mathbb{R}^1$ (CRAWSHAW, 2020).

To avoid those drawbacks, some authors proposed combining the gradients of tasks instead of the loss functions directly. By doing this, the task-specific parameters can be updated by task-specific gradients, and only the shared parameters are updated by the combination of the gradients. The main challenge of this approach is to define how the gradients of different tasks should be combined. Since the tasks share the same parameters in some layers and compete for the same resources (e.g., convolutional kernels from the shared encoder), this combination must be done by finding a trade-off between them so that shared layers can learn relevant features to all tasks (CIPOLLA; GAL; KENDALL, 2018; NAKAMURA; GRASSI; WOLF, 2021). It can be seen in Figure 7, in which Cipolla, Gal and Kendall (2018) showed that the performance of trained models are sensitive to the importance given to each task. In this study, Cipolla, Gal and Kendall (2018) analyzed the influence of the weighting coefficient by training different models to simultaneously learn two tasks. When semantic segmentation and depth estimation are jointly trained (Figure 7a), the performance for both tasks is worse when depth estimation receives more importance during training. On the other hand, the opposite happens in instance segmentation, as can be seen in Figure 7b, the models' performance is better if depth estimation receives greater importance.



(a) *Semantic segmentation and depth estimation* (b) *Instance segmentation and depth estimation.*

Figure 7 – Performance comparison of different tasks in relation to the variation of weighting coefficients. Source: Cipolla, Gal and Kendall (2018).

The problem is that the linear combination of gradients does not allow finding an appropriate balance between tasks (SENER; KOLTUN, 2018; LIN *et al.*, 2019). As an alternative, Sener and Koltun (2018) proposed to model MTL as a MOO problem to find a solution that avoids any task to dominate any other, i.e. a Pareto optimal solution. For this, the authors proposed to minimize the norm of the convex combination of the gradients as an upper bound of the optimization problem. By doing this, they showed that this method can achieve better results than previous methods of simple linear combination and its variations. However, the variations in the gradients' scale are still a problem, which can significantly impact the final solution. Since the method considers the minimum norm of the convex combination of the gradients, it may end up prioritizing only the tasks that have lower gradient norms and not learning the other tasks well. To find a solution that considers the tasks equally important, Harada, Sakuma and Kobayashi (2006), Katrutsa *et al.* (2020) remove the scale influence by normalizing the gradients for each task. In Mao *et al.* (2020), it is proposed to use the Tchebycheff metric to handle non-convex problems. Instead of finding a single optimal solution, Lin *et al.* (2019), Mahapatra and Rajan (2020) propose to find a set of Pareto optimal solutions with different trade-offs between tasks.

3.3 Final Considerations

Comparing the optimization approaches for MTL, the one that models it as a MOO represents the state-of-the-art. Because MOO by gradient descent is an old problem and well studied in several areas (MIETTINEN, 1998; EHRGOTT, 2005; HWANG; MASUD, 1979), one could search for widely used methods in literature and adapt to the MTL problem in neural networks. However, there is a detail that the previous works did not consider and that can directly influence the final results, this is the temporal factor of the learning process. The existing methods consider only the current tasks' gradients to find a common descent direction, not analyzing the previously defined descent directions or any indication if they are being representative to learn all tasks. For this reason, there is a tendency for the solution to prefer a specific direction at every iteration. As mentioned by Vandenhende *et al.* (2022), simply applying an MOO method can assert a feasible common descent direction that allows a Pareto optimal solution, but it does not necessarily mean that the performance of all tasks will be properly improved.

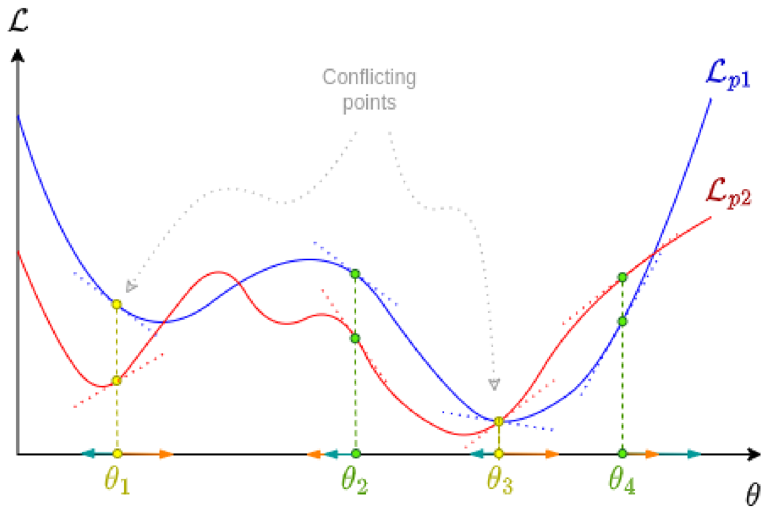
CHAPTER

4

PROPOSED APPROACH

The present work aims to simultaneously learn multiple conflicting tasks, $M \geq 2$, using a single neural network with a hard parameter sharing approach. In this architecture, a subset of hidden layers with low-level features are shared between all tasks, while the rest of the high-level parameters are kept task-specific. Thus, the training process of the shared parameters is performed by optimizing the combination of loss gradients of all tasks. The resultant direction of combining the gradients of all tasks is defined as the common descent direction.

Figure 8 – One-dimensional example of two loss functions and some points where the two tasks are conflicting.



Source: Elaborated by the author.

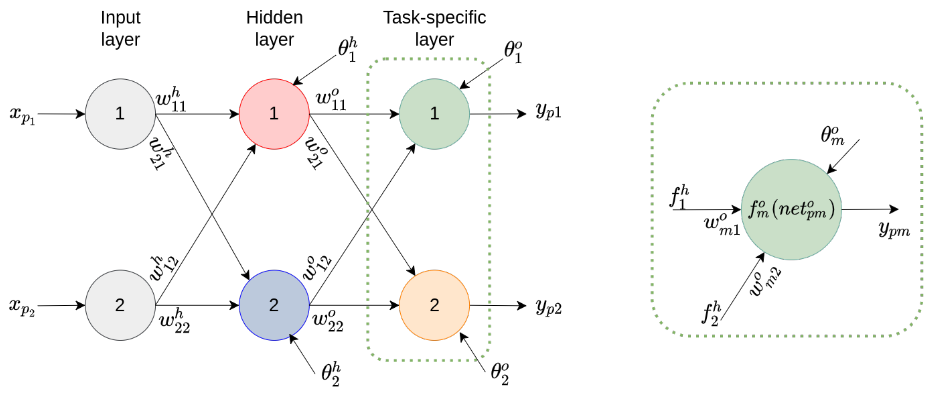
We define two tasks as conflicting when their gradients have different directions at some iterations during the training. For example, consider the one-dimensional loss functions of two tasks with respect to the parameter θ in Figure 8. In this illustration, the gradients of the loss functions are represented as blue and orange arrows at some sampled θ points. It is possible to observe that the gradients can have the same directions at some regions, as in θ_2 and θ_4. When it happens, the descent directions of both functions are the same, benefiting the convergence of both loss functions. However, the gradients can have opposite directions at some other regions, as in θ_1 and θ_3. In this case, the descent direction of a task is not representative for the other, making the tasks conflict with each other. This is because optimizing the shared network parameter in relation to one task can negatively impact the performance of the other. Expanding to a larger scenario with more than two tasks and more parameters, this idea of conflicting tasks is also valid. If at some iterations the method chooses an inappropriate common descent direction, it can negatively impact the performance of some tasks. Therefore, the proposed work focuses on the combination of gradient vectors from different conflicting tasks in an architecture based on the hard parameter sharing approach. To achieve this goal, we modeled and validated the multi-task learning considering the temporal behavior of the gradients to create a dynamic bias that guarantees better performances for all tasks without giving up the optimality condition of multi-objective optimization. This chapter presents the proposed approach and method details.

4.1 Problem Definition and Formulation

To learn multiple tasks we consider an architecture with hard parameter approach due to its simplicity and computational efficiency compared to the soft parameter sharing. As presented in Section 2, Multi-task Learning (MTL) in this approach is commonly formulated as a linear combination of loss functions, $\sum k_m \mathscr{L}_m$, $\forall m = 1, \ldots, M$ (Equation 2.1), where the coefficient k_m defines the importance of each task during training iteration.

We assume to learn conflicting tasks of different natures that are closely related to each other. Therefore, although one task helps in the learning process of the other, we must deal with variations in the order of gradients' magnitude and its directions to properly balance the importance of each task during training. Otherwise, the common descent direction defined to optimize the shared parameters may not be representative for all tasks and the final model's

Figure 9 – Example of an MLP with two output neurons to learn Task 1 and Task 2.



Source: Elaborated by the author.

performance can be jeopardized.

To formulate the problem, consider a Multilayer Perceptron (MLP) with N shared neurons in the hidden layer and M task-specific neurons in the output layer to learn the tasks $y_m, \forall m \in M$ (Figure 9). For simplicity, we will assume that all activation functions are linear. Refer to Haykin (2009), Mello and Ponti (2018) for a more comprehensive understanding of MLP formulation.

A task-specific neuron in the output layer of a task m can be represented as:

$$\begin{aligned} net^o_{pm} &= \sum_{n=1}^{N} [f^h_n(net^h_{pn}) \cdot w^o_{mn}] + \theta^o_m, \\ \hat{y}_{pm} &= f^o_m(net^o_{pm}). \end{aligned} \tag{4.1}$$

where n is the index of a neuron in the hidden layer (see Figure 9).

Therefore, given an input data $\mathbf{x}_p = [x_{p_1}, x_{p_2}]$ from a sample of the set $p \in P$, in the naive MTL, the objective function $\mathscr{L}_p$ for M tasks is defined as the simple sum of each task loss:

$$\mathscr{L}_p = \sum_{m=1}^{M} k_m \mathscr{L}_{pm} = k_m \sum_{m=1}^{M} (y_{pm} - \hat{y}_{pm})^2,$$

where $k_m = 1$ and $\hat{y}_{pm}$ is the annotation of task m given an input data p.

Focusing on a single neuron in the hidden layer, the update of the weight w^h_{11} will be:

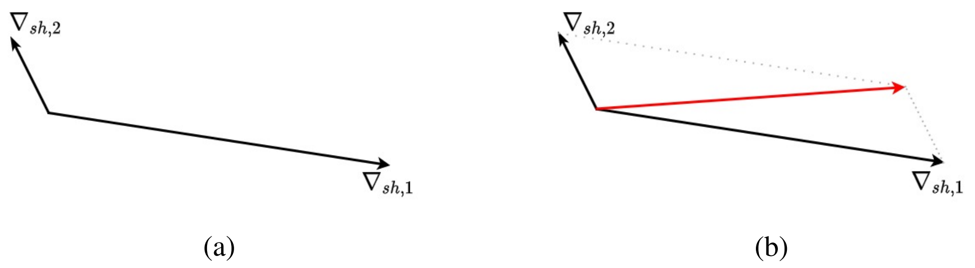
$$w^h_{11}(t+1) = w^h_{11}(t) - \eta \frac{\delta \mathscr{L}_p}{\delta w^h_{11}}.$$

Rewriting, we can also represent the partial derivative as:

$$\begin{aligned} \frac{\delta \mathscr{L}_p}{\delta w^h_{11}} &= \frac{\delta \mathscr{L}_{p1}}{\delta w^h_{11}} + \frac{\delta \mathscr{L}_{p2}}{\delta w^h_{11}} \\ &= \left[2(y_{p1} - \hat{y}_{p1}) \cdot \overbrace{\frac{\delta f^o_1}{\delta net^o_{p1}}}^{1} \cdot \underbrace{\frac{\delta net^o_{p1}}{\delta f^h_1}}_{w^o_{11}} \cdot \overbrace{\frac{\delta f^h_1}{\delta net^h_{p1}}}^{1} \cdot \underbrace{\frac{\delta net^h_{p1}}{\delta w^h_{11}}}_{x_{p1}} \right] \\ &+ \left[2(y_{p2} - \hat{y}_{p2}) \cdot \overbrace{\frac{\delta f^o_2}{\delta net^o_{p2}}}^{1} \cdot \underbrace{\frac{\delta net^o_{p2}}{\delta f^h_1}}_{w^o_{21}} \cdot \overbrace{\frac{\delta f^h_1}{\delta net^h_{p1}}}^{1} \cdot \underbrace{\frac{\delta net^h_{p1}}{\delta w^h_{11}}}_{x_{p1}} \right]. \end{aligned} \tag{4.2}$$

This naive combination of loss functions can be used to handle multiple tasks. However, we can observe in Equation 4.2 that the terms that most influence the weights update in the

Figure 10 – Common descent directions between two gradient vectors, $\nabla_{sh,1}$ and $\nabla_{sh,2}$, considering the simple sum of loss functions. Best visualized with color



hidden neurons are those related to each task loss. Therefore, when we start to deal with problems of different natures (e.g., classification and regression), we also have to deal with problems of different orders of loss magnitude.

Thus, if $(y_{p1} - \hat{y}_{p1}) \gg (y_{p2} - \hat{y}_{p2})$, when performing the sum of errors, the descent direction will be representative in relation to the problem with the highest loss value, and may cause the descent direction of the other task to be neglected. As a consequence, the simultaneous learning of y_1 and y_2 can be affected. For example, considering a case in which $\mathscr{L}_{p1} \gg \mathscr{L}_{p2}$ and the gradient magnitude is proportional to its loss scale (Figure 10a), performing simple sum without scaling tasks will mostly update the shared parameters, w_{sh}, in relation to Task 1 (Figure 10b). For simplicity, we will denote the gradient vector of the loss function of a Task m w.r.t. the shared parameters by $\nabla_{sh,m}$.

In the literature, a common way to avoid this is to manually define weighting coefficients to scale each task, defining the loss for M output neurons as:

$$\mathscr{L}_p = \sum_{m=1}^{M} k_m (y_{pm} - \hat{y}_{pm})^2. \tag{4.3}$$

Then, the update of the weight w_{11}^h in the hidden neuron will be:

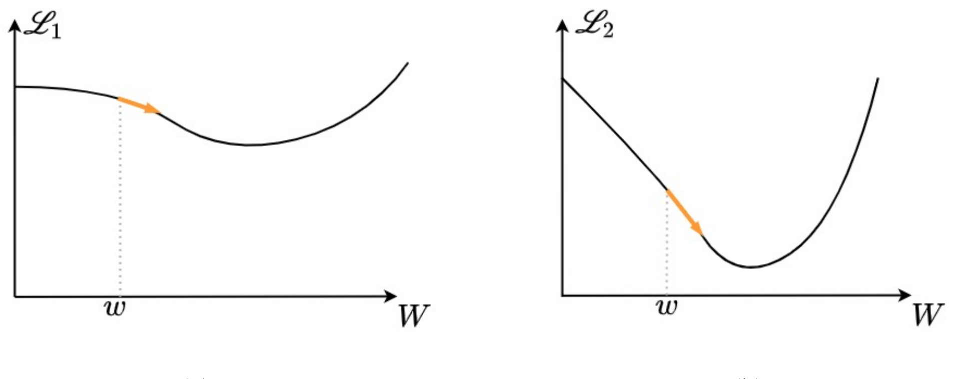
$$\frac{\delta \mathscr{L}_p}{\delta w_{11}^h} = k_1 \frac{\delta \mathscr{L}_{p1}}{\delta w_{11}^h} + k_2 \frac{\delta \mathscr{L}_{p2}}{\delta w_{11}^h},$$

and the update of all shared weights is defined by:

$$\frac{\delta \mathscr{L}_p}{\delta \mathbf{w}_{sh}} = k_1 \frac{\delta \mathscr{L}_{p1}}{\delta \mathbf{w}_{sh}} + k_2 \frac{\delta \mathscr{L}_{p2}}{\delta \mathbf{w}_{sh}},$$

where $\frac{\delta \mathscr{L}_{p1}}{\delta \mathbf{w}_{sh}}$ and $\frac{\delta \mathscr{L}_{p2}}{\delta \mathbf{w}_{sh}}$ are the partial derivative of the loss functions w.r.t. the shared parameters:

Figure 11 – One-dimensional example of loss functions, $\mathscr{L}_1$ and $\mathscr{L}_2$, for different weights.



(a) (b)

Source: Elaborated by the author.

$$\frac{\delta \mathscr{L}_{p1}}{\delta \mathbf{w}_{sh}} = \left[\frac{\delta \mathscr{L}_{p1}}{\delta w^h_{11}}, \frac{\delta \mathscr{L}_{p1}}{\delta w^h_{12}}, \frac{\delta \mathscr{L}_{p1}}{\delta w^h_{21}}, \frac{\delta \mathscr{L}_{p1}}{\delta w^h_{22}}, \frac{\delta \mathscr{L}_{p1}}{\delta \theta^h_{1}}, \frac{\delta \mathscr{L}_{p1}}{\delta \theta^h_{2}} \right], \tag{4.4}$$

$$\frac{\delta \mathscr{L}_{p2}}{\delta \mathbf{w}_{sh}} = \left[\frac{\delta \mathscr{L}_{p2}}{\delta w^h_{11}}, \frac{\delta \mathscr{L}_{p2}}{\delta w^h_{12}}, \frac{\delta \mathscr{L}_{p2}}{\delta w^h_{21}}, \frac{\delta \mathscr{L}_{p2}}{\delta w^h_{22}}, \frac{\delta \mathscr{L}_{p2}}{\delta \theta^h_{1}}, \frac{\delta \mathscr{L}_{p2}}{\delta \theta^h_{2}} \right].$$

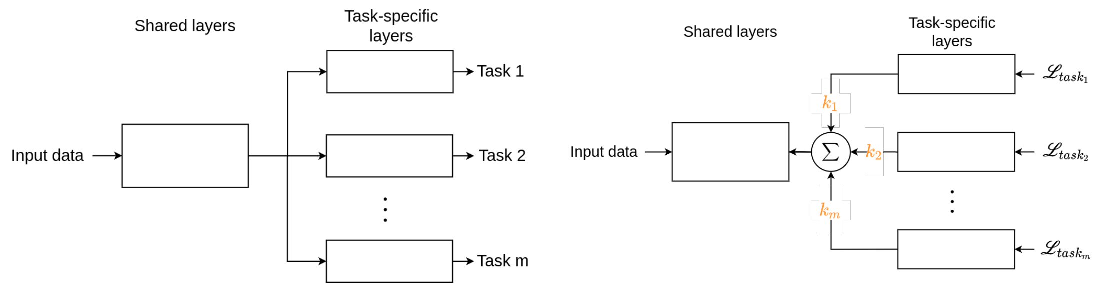
However, defining the importance of each task considering the magnitude of loss function does not allow analyzing the convergence of tasks. This is because each loss function has a behavior and the loss values are not normalized in relation to all tasks. Thus, $\mathscr{L}_{p1} \gg \mathscr{L}_{p2}$ does not mean that Task 1 is converging. An example of this behavior can be observed in Figure 11, in which despite the loss value of Task 1 given weight w is greater than Task 2, its gradient magnitude is smaller than Task 2, $||\nabla_{sh,1}|| \ll ||\nabla_{sh,2}||$. Note that the gradient vector of a task m, $\nabla_{sh,m}$, is the direction with the maximum change given weights w. Therefore, instead of defining the task's importance according to its loss values, we will consider the gradient vectors of each task to find a common descent direction between tasks (Figure 12).

In Sener and Koltun (2018), Désidéri (2012), the common descent direction was defined by searching for weighting coefficients that minimize the norm of the convex combination of the gradients:

$$\underset{k_1,\cdots,k_M}{\operatorname{argmin}} \left\{ \left\| \sum_{m=1}^{M} k_m \nabla_{sh,m} \right\|_2^2 \middle| \sum_{m=1}^{M} k_m = 1, k_m \geq 0 \quad \forall m \right\}. \tag{4.5}$$

This solution will result in the minimal-norm element in the convex hull of the negative gradients. Although the descent direction found by Equation 4.5 is based on an optimality condition (Definition 1), it may not be representative for all tasks when the tasks' losses are not balanced. In this case, the solution will tend to give more preference to the direction with the least norm (red vector in Figure 13).

Figure 12 – Multi-task learning architecture. The output from the shared layers is split into a set of m task-specific layers. During backpropagation, the parameters from task-specific layers are optimized according to the gradients of each task, while in the shared layers, the optimization occurs by combining the gradients of all tasks. This combination is usually performed by a weighted sum, in which each task is associated with a weighting coefficient, k_m.



(a) Feedforward (b) Backpropagation

Source: Elaborated by the author.

4.2 Central Descent Direction

To avoid prioritizing only the direction of a specific task, we remove the scale influence on the problem by considering the unit vector of each task's gradient, $\bar{\nabla}_{sh,m} = \nabla_{sh,m}/\|\nabla_{sh,m}\|$. Therefore, all tasks will be equally important during the descent direction computation. Equation 4.5 can be rewritten as:

$$\underset{k_1,\cdots,k_M}{\operatorname{argmin}} \left\{ \left\| \sum_{m=1}^{M} k_m \bar{\nabla}_{sh,m} \right\|_2^2 \middle| \sum_{m=1}^{M} k_m = 1, k_m \geq 0 \quad \forall m \right\}. \tag{4.6}$$

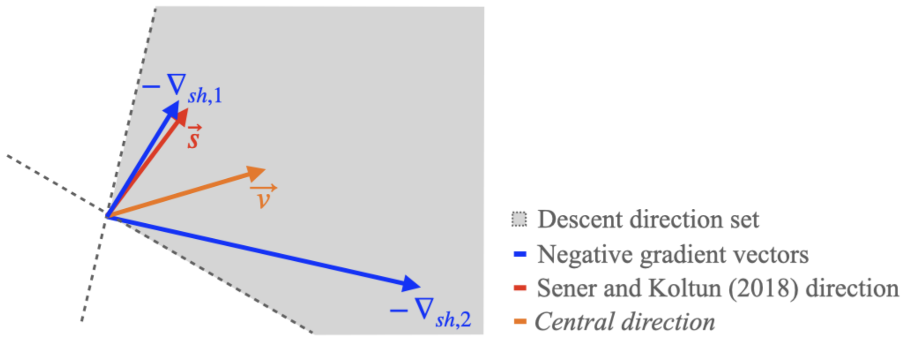
Solving optimization problem (4.6), the common descent direction can be obtained by a convex combination of the normalized gradients of each task with their respective weighting coefficients, $\sum_{m=1}^{M} k_m \bar{\nabla}_{sh,m}$. This direction will be close to the central region of the cone formed by the negative gradient vectors of different tasks (orange vector in Figure 13), which will result in the same relative decrease among them. Since this direction was calculated considering unit vectors, we must recover the scale factor so that the resulting vector lies in the convex hull formed by the negative gradient vectors of the tasks.

This can be done using a scale factor γ as proposed in Katrutsa *et al.* (2020). Therefore, the resulting common descent direction, $\vec{v}$, that lies in the central region between tasks can be defined as:

$$\vec{v} = \gamma \sum_{m=1}^{M} k_m \bar{\nabla}_{sh,m}, \tag{4.7}$$

where $\gamma = \left(\sum_{m=1}^{M} \frac{k_m}{\|\nabla_{sh,m}\|} \right)^{-1}$.

Figure 13 – Common descent directions considering the negative gradient vectors of $\nabla_{sh,1}$ and $\nabla_{sh,2}$. Best visualized with color.



Source: Elaborated by the author.

However, note that this direction and all others studied so far create an assumption that the direction found is representative considering only the gradient analysis at a current time, without considering historical information or any indication that the directions defined during the training process were being representative for all tasks. This means that, regardless of the method adopted to compute a common descent direction, the direction found will tend to prefer a specific descent direction (i.e., the direction that results in lesser or greater magnitude, the central direction, etc.). Thus, to enable the correction of the common descent direction during training, we propose to analyze the convergence of the models by tracking the gradients' norm of each task during a period of the last T iterations and adopt the idea of tensioners that pull the common descent direction for the task that is diverging or under-performing.

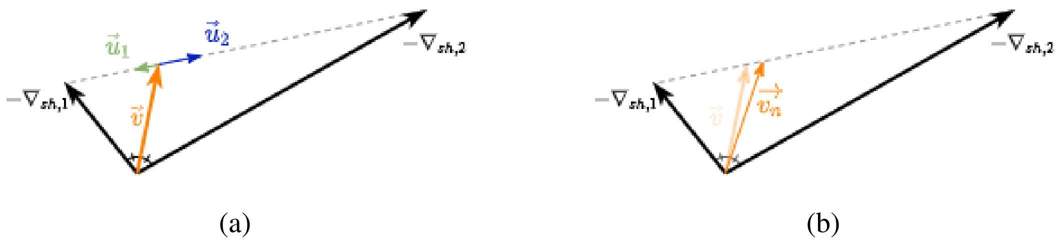
4.3 Task Tensioner

Imagine a rod held vertically on a horizontal surface. Then, two rubber bands are tied to the top end of the rod, and they are pulled in opposite directions. When the two rubber bands are stretched at the same amount, the rod will remain at rest. However, when we stretch more only one of the rubber bands, the tension generated in this band will make the rod lean to its side. This is the general idea of task tensioner, in which the vertical rod can be considered as the common descent direction and the rubber bands pulling the rod represent how bad the learning progress of each task is. Therefore, the more a task starts to diverge, the more tension the rubber band will need to pull the task. By doing this, we encourage the network to regulate the priority level of each task by avoiding performance degradation.

Hence, given a common descent direction, in this case, the central direction, $\vec{v}$, obtained through the Equation 4.7, the tensions of each task can be computed according to the following equation:

$$u_m = c_m \cdot \frac{\nabla_{sh,m} - \vec{v}}{\|\nabla_{sh,m} - \vec{v}\|}, \quad \forall m = 1, \ldots, M, \tag{4.8}$$

Figure 14 – Method for changing the common descent direction between tasks. (a) Tensioners that will pull the common descent direction for the diverging task. (b) New defined descent direction.



Source: Elaborated by the author.

where $||\cdot||$ is l2-norm, and the tension factor, c_m, measures how much each task must be pulled to correct the common descent direction considering a relative variation of gradients' norm during a predefined number of iterations (Figure 14). To obtain this relative variation, the average of gradients' magnitude accumulated during a period of T iterations is calculated by:

$$\zeta_m(t) = \frac{1}{T}\sum_{i=0}^{T-1} \left\|\nabla_{sh,m}(t-i)\right\|. \tag{4.9}$$

Then, the relative variation of the gradient's norm can be obtained by the ratio between current and previous accumulated gradient:

$$\delta_m = \frac{||\zeta_m(t)||}{||\zeta_m(t-1)||} + \log_{10}(\mathscr{L}_m). \tag{4.10}$$

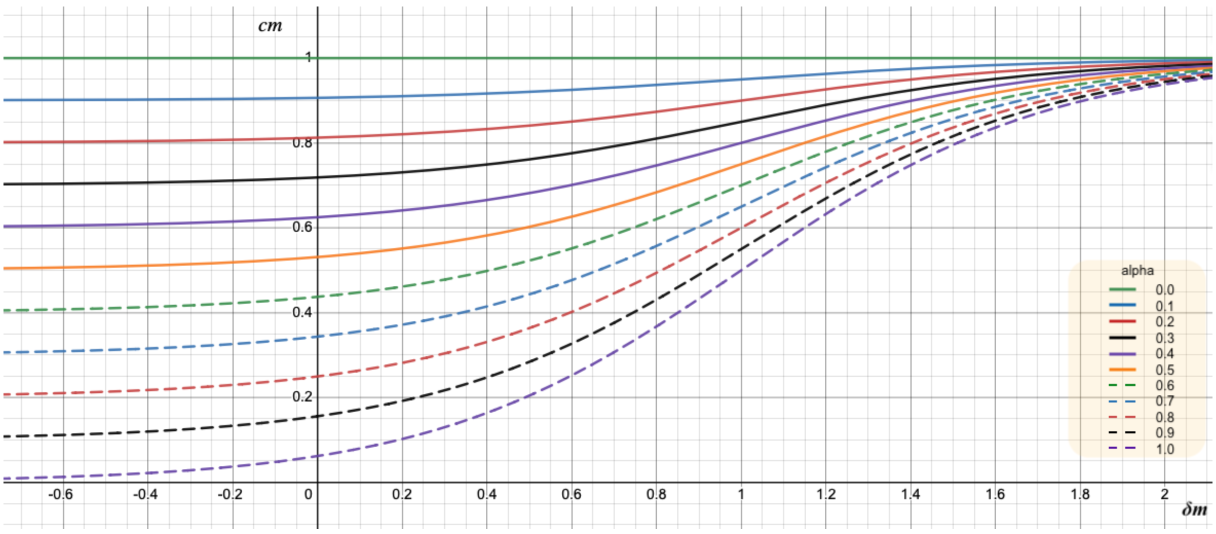
The term $\log_{10}(\mathscr{L}_m)$ aims to penalize tasks that have higher loss, increasing the tension factor of the respective tasks. Despite the fact we used the average of current and previous accumulated gradients, it is worth noting that one could use any other function instead of the common average, as long as this function can effectively abstract the behavior of the gradient variation.

To get how much the common descent direction should be changed given the divergence indicator (δ_m), the tension factor of each task is computed by:

$$c_m = \frac{\alpha}{1+e^{(-\delta_m \cdot e + e)}} + 1 - \alpha. \tag{4.11}$$

Note that the tension factor, c_m, is bounded by the sigmoid curve (Figure 15). Therefore, this functional form of Equation 4.11 controls the tensor factor in two circumstances. The first situation is to prevent the tensioner from pulling too much the common descent direction in case of a big increase in the magnitude of one task (i.e., very big δ_m), otherwise, the algorithm would be allowing this task to dominate over the others. The second situation is to prevent the tensioner

Figure 15 – Behavior of the tensor factor w.r.t. the relative variation of the gradient's norm given the α values (Equation 4.11).



Source: Elaborated by the author.

from pushing the descent direction when a task has a very small loss (i.e., very negative δ_m), otherwise, the algorithm would be allowing this task to increase its loss instead of minimizing it.

Moreover, since we want to change the descent direction only if some task is diverging, the c_m values are calculated so that the direction does not change too much from the convergence region of all tasks. Therefore, to regulate the sensitivity of the tension factor, a constant $\alpha \in [0,1]$ is applied to Equation 4.11. Hence, if $\alpha = 0$, the resulting vector will be exactly the vector $\vec{v}$. And, the higher the α value, the more the common descent direction will be pulled towards the task that is underperforming or diverging.

The new common descent direction among tasks can be defined by the sum of the current common descent direction, $\vec{v}$, with the linear combination of each task's tension (Equation 4.8):

$$\vec{v}_n = \vec{v} + \sum_{m=1}^{M} c_m \cdot \frac{\nabla_{sh,m} - \vec{v}}{\|\nabla_{sh,m} - \vec{v}\|}, \tag{4.12}$$

subject to $\nabla_{sh,m} \cdot \vec{v}_n \geq 0, \forall m = 1, \ldots, M$.

Algorithm 1 shows the proposed method that takes into account gradient history during training to pull the common descent direction for the diverging task.

4.4 Final Considerations

This chapter presents the proposed method to analyze the convergence of tasks during the training process to dynamically adapt the common descent direction and find a trade-off between conflicting tasks that maximize simultaneous learning. By monitoring the training progress, we can define a new descent direction that prevents the model from learning more about a specific task by pulling tasks that are diverging or that are not being benefited during training. To isolate the architectural factor and focus the contribution on the optimization method, this work employs the hard parameter sharing architecture due to its simplicity and lower computational cost.

Algorithm 1 – Gradient accumulation

```
1: for $i = 0, ..., I-1$ do
2:     for $m = 1, \cdots, M$ do
3:         $\mathcal{L}_m \leftarrow \mathcal{L}(f_m(x_i; \mathbf{w}_{sh}, \mathbf{w}_m), y_{i,m})$        ▷ Compute task-specific loss
4:         $\nabla_m \leftarrow \frac{\partial \mathcal{L}_m}{\partial \mathbf{w}_m}$        ▷ Gradient descent on task-specific parameters
5:         $\nabla_{sh,m} \leftarrow \frac{\partial \mathcal{L}_m}{\partial \mathbf{w}_{sh}}$        ▷ Gradient descent on shared parameters
6:     end for
7:     Solve (4.6) to find $\vec{v}$        ▷ Common descent direction (KATRUTSA et al., 2020)
8:     if $i \geq T$ then
9:         Compute new common descent direction using Algorithm 2.
10:    else
11:        $\vec{v}_n \leftarrow \vec{v}$
12:    end if
13:    Update $\mathbf{w}_{sh}$ w.r.t $\vec{v}_n$ with chosen optimizer;
14:    Update $\mathbf{w}_m$, $\forall m$ with chosen optimizer.
15: end for
```

Algorithm 2 – Compute task tension

```
Require: $i \geq T$
1: for $m = 1, \cdots, M$ do
2:     Compute relative change, $\delta_m$ (4.10)
3:     Compute tension factor, $c_m$ (4.11)
4:     Compute task tension, $u_m$ (4.8)
5: end for
6: $\vec{v}_n \leftarrow \vec{v} + \sum_{m=1}^{M} u_m$        ▷ Update common descent direction (4.12)
7: return $\vec{v}_n$
```

CHAPTER 5

EXPERIMENTS

In this chapter, sensitivity analysis and quantitative evaluations with different baselines are performed to compare the proposed method. To analyze the behavior of different descent direction methods, and ensure that it is not influenced by the network architecture or other variables, we adopted a Multi-layer Perceptron (MLP) in the sensitivity analysis to learn problems from multiple logical operators. Afterward, the evaluation is performed in an image domain application considering the classification and reconstruction problems in the MNIST dataset adapted to perform multi-task learning. Moreover, we validated our method on the scene understanding problem, to learn the tasks of semantic segmentation, instance segmentation, and depth estimation using the CityScapes Dataset.

5.1 Experimental Setup and Metrics

The experiments were conducted using a machine with Intel Core i7-6700, 32GB RAM, and RTX 2080Ti 11GB. All implementations were made using the Pytorch framework, and the architecture adopted in this project was the encoder-decoder with a hard parameter sharing approach. Therefore, the first part of the architecture (encoder) will have the purpose of extracting general features that benefit the learning of tasks. And in the second part, the architecture will be divided so that each task has one decoder and the backpropagation is performed only between the parameters of each task, allowing the extraction of more specific features.

For quantitative evaluations, we used the mean squared error (MSE), root mean squared error (RMSE), classification accuracy, and mean intersection-over-union (mIoU):

$$
\begin{aligned}
MSE &= \frac{1}{N}\sum_{n=1}^{N}\left(y_n - \hat{y}_n\right)^2,\\
RMSE &= \sqrt{\frac{1}{N}\sum_{n=1}^{N}\left(y_n - \hat{y}_n\right)^2},\\
Accuracy &= \frac{TP+TN}{TP+FP+FN+TN},\\
IoU_c &= \frac{y \cap \hat{y}}{y \cup \hat{y}} = \frac{\sum_{n=1}^{N}\min(y_n,\hat{y}_n)}{\sum_{n=1}^{N}\max(y_n,\hat{y}_n)}, \quad (y_n,\hat{y}_n) \in \{0,1\},\\
mIoU &= \frac{1}{C}\sum_{c=1}^{C} IoU_c
\end{aligned}
$$

where y is the estimated value, $\hat{y}$ is the ground truth, and N is the total number of pixels in the evaluated images. TP, TN, FP, and FN are the numbers of true positive, true negative, false positive, and false negative classifications, respectively. Lastly, C is the number of segmentation classes.

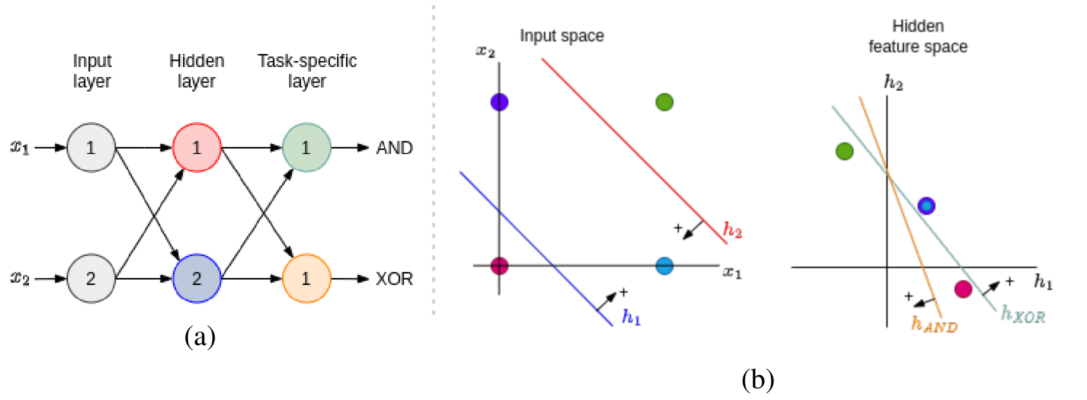
5.2 Multiple Logical Operators

In this experiment, we adopted to evaluate the simultaneous learning of the logical operators XOR and AND because we know exactly how many neurons are needed to generate the hyperplanes that separate the sets of tasks' data in the feature space. Therefore, we eliminate external influences that generate uncertainty if it was not possible to obtain a better result due to the architecture or problem modeling, allowing us to evaluate only the descent direction methods. Furthermore, the problems are both related and conflicting with each other, since in shared neurons, the AND task will try to separate the feature space into two regions, while the XOR task will need to separate the feature space into three regions. Figure 16a shows a MLP with two shared neurons and two task-specific output neurons. The hyper-planes required to separate the data of both tasks are shown in Figure 16b.

The experiments were performed considering the common descent directions computed according to the methods of uniform weighting, minimum-norm (SENER; KOLTUN, 2018), central direction (KATRUTSA *et al.*, 2020), and gradient modification with tasks' tensioners (proposed). To analyze the descent methods considering tasks with different orders of magnitude, the annotations of XOR task were multiplied by a constant $c \geq 1$ to change the problem's scale.

Each model was trained using mean squared error (MSE) loss function and stop condition with the limit of 3,000 epochs or an error smaller than 1e−3. Since training results can vary from one run to another, each method was trained 50 times with a different seed value to get deterministic random data. Furthermore, the models were trained considering a set $lr = \{5\text{e}{-5}, 1\text{e}{-4}, 5\text{e}{-3}, 1\text{e}{-3}, 5\text{e}{-2}, 1\text{e}{-2}\}$ of learning rates and the Stochastic Gradient Descent

Figure 16 – (a) MLP for AND and XOR tasks. (b) Feature space and hyper-planes necessary to separate the input data. Best visualized with color.



Source: Elaborated by the author.

(SGD) optimization algorithm with momentum. Thus, the reported results are the models that best separated the input data.

5.2.1 Sensitivity Analysis

The Influence of T and α: To verify the behavior of using tensioners in relation to the values of α and T, we conducted this experiment considering the periods of $T = \{5, 10, 15\}$ and varying the α values between the interval $[0, 1]$ with a step of 0.1 to adjust the central direction defined by Equation 4.7. The period T defines how much information from the iterations will be used to analyze the convergence of the models. Intuitively, if the value of T is too small, the information analyzed may not be enough to extract the behavior of the gradients, making them more susceptible to noise. On the other hand, by adopting a very large value of T, we may be analyzing more information than necessary.

Table 1 – Average performance without changing the problem's scale. Results show the effect of using the proposed method to adjust the central direction considering different periods of iterations, T, and α values. In the case of $\alpha = 0.0$, there is no adjustment of the common descent direction.

	$T = 5$		$T = 10$		$T = 15$	
α	Training epoch avg.	Convergence rate	Training epoch avg.	Convergence rate	Training epoch avg.	Convergence rate
0.0	574.44 ± 43.37	100.00	574.44 ± 43.37	100.00	574.44 ± 43.37	100.00
0.1	539.62 ± 30.39	100.00	527.46 ± 27.57	100.00	535.64 ± 27.89	100.00
0.2	496.52 ± 26.45	100.00	477.62 ± 23.09	100.00	489.36 ± 23.59	100.00
0.3	472.22 ± 27.73	100.00	443.02 ± 21.14	100.00	455.48 ± 22.58	100.00
0.4	456.06 ± 30.84	100.00	415.60 ± 19.14	100.00	428.38 ± 20.66	100.00
0.5	448.42 ± 41.63	100.00	396.08 ± 18.05	100.00	407.20 ± 18.53	100.00
0.6	444.48 ± 57.03	100.00	381.98 ± 17.90	100.00	391.82 ± 17.53	100.00
0.7	437.26 ± 54.05	100.00	370.00 ± 16.41	100.00	379.72 ± 15.42	100.00
0.8	430.52 ± 47.62	100.00	359.28 ± 15.90	100.00	369.58 ± 15.84	100.00
0.9	430.32 ± 66.24	100.00	358.68 ± 15.42	100.00	360.36 ± 14.83	100.00
1.0	431.48 ± 87.40	100.00	343.46 ± 15.22	100.00	362.64 ± 14.76	100.00

Table 2 – Average performance considering $c = 10$ for XOR task. Results show the effect of using the proposed method to adjust the central direction considering different periods of iterations, T, and α values. In the case of $\alpha = 0.0$, there is no adjustment of the common descent direction.

	T=5		T=10		T=15	
α	Training epoch avg.	Convergence rate	Training epoch avg.	Convergence rate	Training epoch avg.	Convergence rate
0.0	328.50 ± 25.45	100.00	328.50 ± 25.45	100.00	328.50 ± 25.45	100.00
0.1	235.18 ± 37.23	100.00	253.22 ± 23.56	100.00	247.98 ± 32.79	100.00
0.2	207.06 ± 39.38	100.00	204.09 ± 23.64	100.00	222.64 ± 31.29	100.00
0.3	188.04 ± 36.16	100.00	188.20 ± 24.29	100.00	210.72 ± 38.15	100.00
0.4	177.92 ± 38.25	100.00	181.44 ± 24.26	100.00	207.52 ± 40.53	100.00
0.5	168.66 ± 38.07	100.00	180.64 ± 37.88	100.00	202.02 ± 44.57	100.00
0.6	171.90 ± 46.16	100.00	180.90 ± 75.41	100.00	194.80 ± 37.25	100.00
0.7	183.46 ± 85.43	100.00	193.42 ± 90.41	100.00	195.46 ± 45.51	100.00
0.8	231.50 ± 401.67	96.00	178.30 ± 65.38	98.00	252.32 ± 209.94	98.00
0.9	341.10 ± 609.80	96.00	232.08 ± 198.64	94.00	266.84 ± 166.23	96.00
1.0	340.16 ± 574.46	94.00	231.94 ± 177.96	98.00	361.94 ± 275.57	98.00

Regarding α values, it aims to control the sensitivity of the tensor factor. Thus, if $\alpha = 0.0$, no tension force is applied, and the common descent direction will be the direction defined by Equation 4.7. On the other hand, the higher the α value, the more the common descent direction will be pulled towards the task that is underperforming or diverging (Figure 15).

Table 1 presents the results obtained for the problem of logical operators XOR and AND with similar gradient magnitudes ($c = 1$). In contrast, Table 2 presents the results obtained for the same logical problems, but with gradients in different orders of magnitude ($c = 10$). Training epoch avg. represents the number of epochs necessary for the model to converge, while the convergence rate is the ratio between the number of models that converged among the 50 performed training.

We can observe from the results presented in both tables that the number of training epochs needed to converge tends to show less variance as the value of T increases. However, when conditioning T to a larger number, e.g., $T = 15$, the model convergence becomes slower.

Regarding the variation of the α value, we can observe in Table 1 that the greater the influence of the tensor factor, the faster the convergence. However, when we have tasks with different gradient magnitudes, the α value tends to slow down the convergence when it is close to 0.0 or 1.0. Additionally, because of the penalty term in Equation 4.10, the task with greater loss will have more strength than the others until its loss becomes as small as the other tasks. To regulate this phenomenon, the influence of the tensioner can be moderated by setting an intermediate value of α. Therefore, considering the analysis presented in this study, we set $T = 10$ (Equation 4.3) and $\alpha = 0.3$ (Equation 4.11) to perform the following experiments presented in this section.

Table 3 – Average performance of evaluated methods without changing the problems' scale.

Descent direction method	Convergence rate (%)	Training epochs average
Single AND	100.00	351.70 ± 2.62
Single XOR	100.00	871.20 ± 52.95
Uniform weighting	100.00	484.86 ± 18.12
Min-norm	100.00	758.56 ± 137.15
Central dir.	100.00	574.44 ± 43.37
Min-norm + tensor	100.00	493.16 ± 27.94
Central dir. + tensor	100.00	$\mathbf{443.02 \pm 21.14}$

5.2.2 Performance Comparison

Similar gradients' magnitude: Table 3 presents the average performance of each method considering $c = 1$, i.e., without much difference in the gradients' magnitude. The first two rows correspond to the models separately trained for each task, which we define as our baseline.

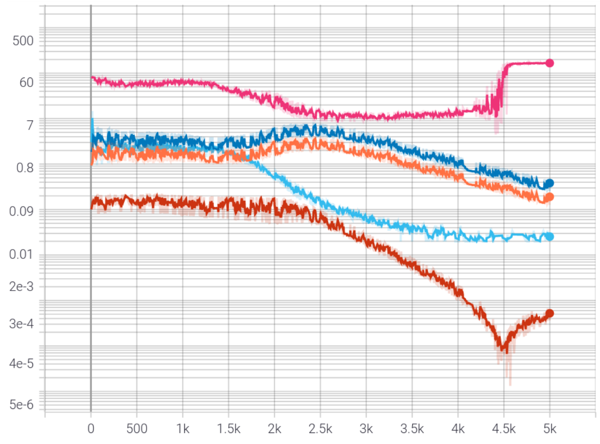
As can be seen, all methods were able to create hyperplanes capable of separating the data for both tasks without significant differences in the convergence time. With the exception of the method proposed by Sener and Koltun (2018), which has a slightly slower convergence compared to other methods because it prioritizes the task direction with a lower gradient magnitude.

Regarding the use of the tensioners proposed in the present work to change the common descent direction according to each task's learning progress, the performance must be similar to or better than the unaltered directions. As can be seen, the proposed method resulted in faster convergence. In the case of the central direction, the use of tensioners reduced the number of epochs needed from 574.44 to 443.02 epochs, resulting in the method that obtained the fastest convergence. And when we apply the tensioners in the minimum-norm method, which was the method that had the slowest convergence, it was possible to reduce from 758.56 to 493.16 epochs.

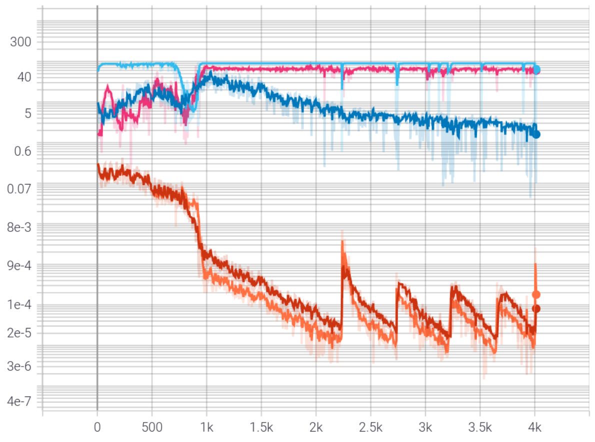
Different gradient's magnitude: To analyze the behavior of descent direction methods considering problems closer to which the proposed method seeks to solve, i.e., when we have tasks with variation in the gradients' magnitude due to the tasks having different types of nature or even loss functions, the annotations of the XOR task were multiplied by the constant $c = 10$ to simulate this variation.

The behavior of each descent direction method can be observed in the learning curves of Figure 17. When uniform weighting is used, we can see that the common descent direction gives preference to the task with greater magnitude, in which the angle between the common descent direction and the XOR task is close to zero (light blue curve in Figure 17a). This can make simultaneous learning to be substantially accomplished in relation to a single task, and the common descent direction does not be feasible for the other tasks. As in the case of the AND task, where in some iterations the angle between its direction and the defined common descent

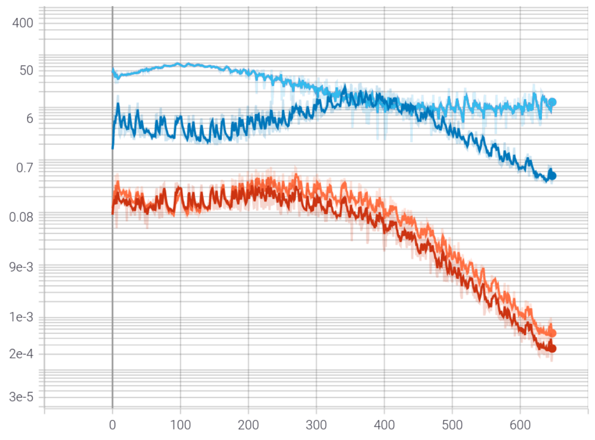
Figure 17 – Learning curves of different descent direction methods. Pink and light blue curves represent the angles of AND and XOR tasks in relation to the common descent direction, respectively. Red and dark blue curves represent the gradients' norm of AND and XOR tasks, respectively. Finally, orange curves represent the gradient's norm of the common descent direction.



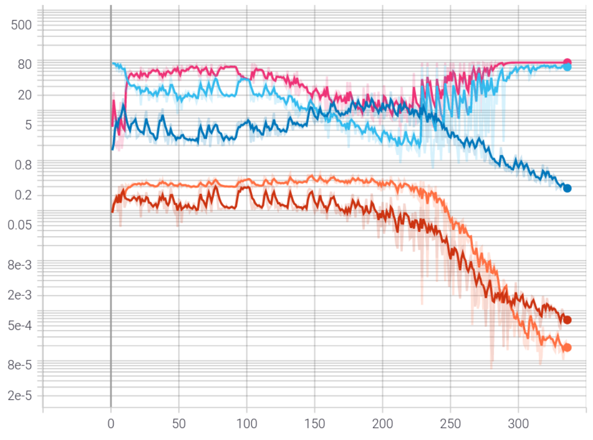
(a) Uniform



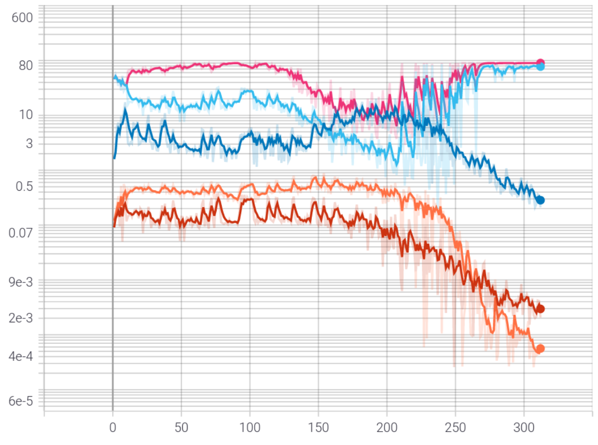
(b) Minimum-norm



(c) Central direction



(d) Minimum-norm + tensor



(e) Central direction + tensor

Source: Elaborated by the author.

direction is greater than 90 degrees (Definition 2.3). On the other hand, when we have tasks with differences in the gradient scale, the method proposed by Sener and Koltun (2018) gives preference to the gradient direction with the lowest norm, making the model converge faster in relation to this task, which can hinder the learning of the other tasks or slow down the learning process (Figure 17b).

The central direction is the direction close to the central region of the cone formed by the negative gradients, resulting in the same angle between tasks (light blue and pink curves are overlapping in Figure 17c). Therefore, this direction tends to give equal importance to all tasks, regardless of the gradients norm. Figures 17d and 17e show the minimum-norm and central directions changed with the use of tensioners, respectively. As it can be seen, in the first iterations the central direction method assigns the same importance to both tasks, resulting in the common descent direction having the same angle between the tasks. However, we can observe that, in addition to the XOR task norm being higher, it increases significantly compared to another task (dark blue curve in Figure 17e). From $T = 10$ iterations, the use of tensioners starts to dynamically change the common descent direction in relation to the task that is diverging. Thus, as soon as the norm of both tasks starts to reduce simultaneously, the common descent direction starts to give practically the same importance to the tasks (after approximately 230 iterations), with a small variation due to the penalty factor in Equation 4.10. The same behavior can be observed when we apply the use of tensioners in the direction of the minimum-norm method, proving that the proposed method can adapt to different descent directions.

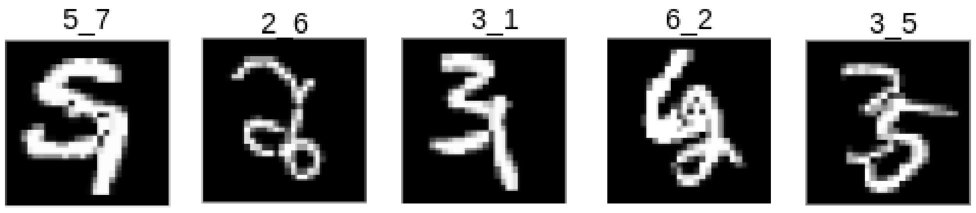
The average of the 50 trainings is shown in Table 4. Since only the scale of the XOR problem was changed, the performance of the model trained separately for the AND task remains the same. In this experiment, the impact of using different descent directions methods becomes more visible. For the single-task XOR, the model needs more iterations to converge when the scale's problem changes. When performed the simultaneous learning of both tasks, the uniform weighting resulted in the worst performance, being able to separate the feature space only in 46% of the 50 trainings, and, in the best case, needing more than 1,000 epochs for convergence. Regarding the methods of central direction and minimum-norm (SENER; KOLTUN, 2018; KATRUTSA *et al.*, 2020), both converged in all trainings. However, as can be seen in the fourth

Descent direction method	Convergence rate (%)	Training epochs average
Single-task AND	100.00	351.70 ± 2.62
Single-task XOR	100.00	$2{,}979.00 \pm 0.80$
Uniform weighting	46.00	$1{,}831.60 \pm 810.81$
Min-norm	100.00	$1{,}998.44 \pm 316.38$
Central dir.	100.00	328.50 ± 25.45
Min-norm + tensor	100.00	210.38 ± 14.96
Central dir. + tensor	100.00	$\mathbf{188.20 \pm 24.29}$

Table 4 – Average performance of evaluated methods considering $c = 10$ for XOR task.

row of Table 4, the method proposed by Sener and Koltun (2018) needs more iterations. Finally, when we use the tensioners to change the central and the minimum-norm direction, we again improve the convergence of the models, significantly reducing the number of epochs compared to the minimum norm method and obtaining the best performance among the evaluated methods.

Figure 18 – Images examples from the MNIST dataset modified for multi-task learning. Images have a size of 28×28 pixels.



Source: Elaborated by the author.

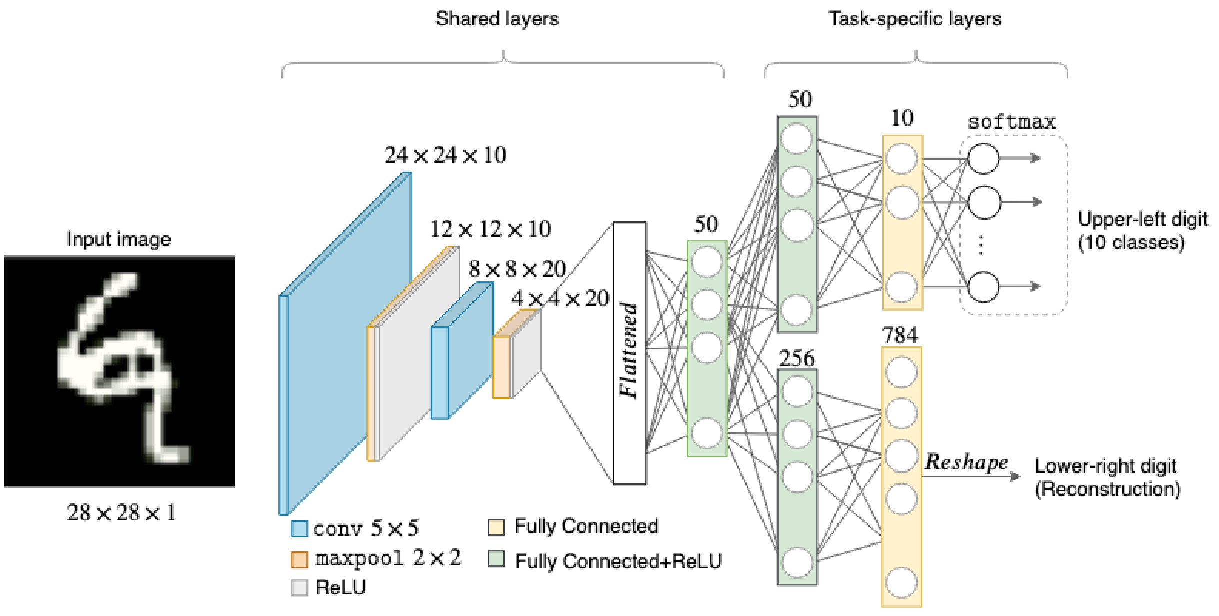
5.3 Handwritten Digit Problem

In order to validate that the proposed method is independent of a specific application, we performed the experiment in the image domain considering the dataset of handwritten digits (LECUN *et al.*, 1998). To turn it into a multi-task learning problem, we use the idea of adapting the MNIST dataset (SABOUR; FROSST; HINTON, 2017; SENER; KOLTUN, 2018), in which each image is composed by the overlapping of two digit images (Figure 18). For this purpose, two randomly selected 28x28 images were placed in the upper-left and lower-right corners of a 32x32 image. Then, the resulting image was resized to 28x28. The training and validation images were generated by splitting the train set images, and the test images using the test set from the MNIST dataset. A total of 50,000 training images, 10,000 validation images, and 10,000 test images were generated.

In the original proposal, the problem is to classify the digits in the upper left and lower right corners. However, since all digits are from the same dataset, note that both tasks have the same nature, changing only the position where they are located in the image. In order to change the scale and nature of the problem, we defined the first task as a classification of the upper-left digit and the second as an autoencoder to reconstruct the lower-right digit. The intuition behind the problem is to keep tasks closely related, but make them conflicting during training. Therefore, while the autoencoder will try to compress the input data into a latent space and reduce the reconstruction error, the other task will try to extract representative features to correctly classify the upper-left digit.

We use the modification of the LeNet architecture shown in Figure 19 (LECUN *et al.*, 1998; SENER; KOLTUN, 2018). The models were trained considering the Stochastic Gradient Descent (SGD) optimization algorithm with momentum and a set $lr = \{1e{-}3,\ 5e{-}3,\ 1e{-}2,$

Figure 19 – Multi-task learning architecture for upper-left digit classification and lower-right digit reconstruction.



Source: Elaborated by the author.

5e−2, 1e−1} of learning rates. Thus, we selected the model that obtained the best validation performance to report the results.

The Negative Log-likelihood (NLL) loss was used for the classification task and the MSE loss for the autoencoder. Each model was trained ten times with a different seed value to get deterministic random data. Furthermore, the training process was interrupted only when there was no improvement in validation metrics during ten epochs.

Table 5 shows the average performance of the models evaluated in the test set. The first row represents the accuracy and reconstruction error of the models separately trained for each task. Therefore, we use them as a baseline to evaluate the multi-task models. As presented in the sensitivity analysis, the common descent direction obtained by the uniform weighting tends to give greater importance to the task with the highest gradient norm. In this experiment, uniform

Table 5 – Average results of the digit classification and reconstruction tasks in the test set. We report the accuracy for classification (higher is better) and MSE for reconstruction task (lower is better).

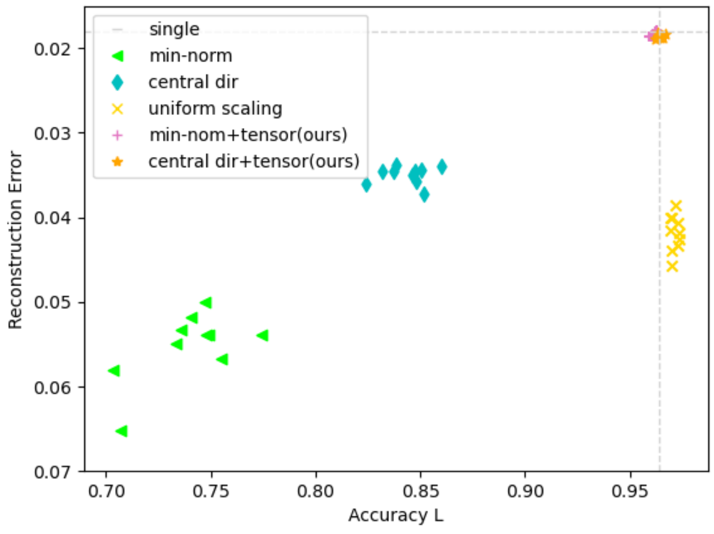
Descent direction method	Upper-left digit Classification accuracy ↑	Lower-right digit Reconstruction error ↓
Single-task	0.9639 ± 0.0024	0.0181 ± 0.0017
Uniform weighting	**0.9714 ± 0.0018**	0.0418 ± 0.0021
Min-norm	0.7393 ± 0.0213	0.0552 ± 0.0042
Central dir.	0.8473 ± 0.0107	0.0350 ± 0.0010
Min-norm + tensor	0.9627 ± 0.0015	**0.0181 ± 0.0005**
Central dir. + tensor	0.9649 ± 0.0015	0.0185 ± 0.0003

weighting was the method that resulted in the best accuracy for the classification problem (second row). However, in addition to extracting relevant features to classification, the shared encoder must be able to compress the input data ($\mathbb{R}^{784}$) into a latent space representation ($\mathbb{R}^{50}$), so that the task-specific decoder can reconstruct this representation. In this case, since the shared encoder was mainly optimized in relation to the classification problem, the uniform weighting reconstruction error was greater than the individual task.

The central direction method (fourth row) obtained a little better result than the minimum-norm method (third row). However, both also had a lower performance than separately training each model.

Finally, the last two rows present the use of tensioners to modify the minimum-norm and central directions. Although the proposed method did not achieve the best performance separately for each task, it resulted in the best balance among the evaluated methods. This can be better observed in Figure 20, which shows the performance of the ten trained models of each method. To simplify the visualization, we inverted the y-axis of the plot. Therefore, the methods with the best results are those closest to the upper-right corner.

Figure 20 – Plot of all trained models on the handwritten classification problem. We use as baseline the average performance of single tasks (gray dashed line). The proposed method resulted in a good balance between tasks, resulting in a performance similar to or even better than single-task models. Upper-right is better.

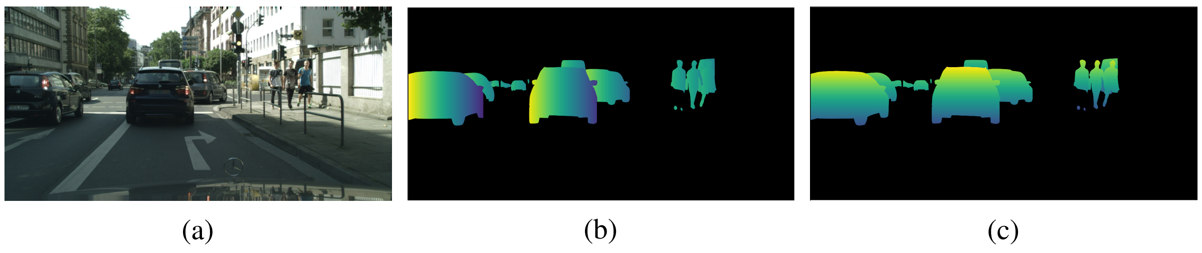


Source: Elaborated by the author.

5.4 Scene Understanding

Finally, we evaluate the proposed method in a more realistic and recent problem, the scene understanding. Therefore, we performed the simultaneous learning of semantic segmentation, instance segmentation, and monocular depth estimation tasks. The training, validation, and test

Figure 21 – The instance segmentation ground truth given an (a) RGB input image. (b) Pixel's displacement at x axis. (c) Pixel's displacement at y axis.



Source: Elaborated by the author.

were performed using the CityScapes dataset (CORDTS *et al.*, 2016), which has images of resolution 1024×2048.

5.4.1 Classes and Annotations

The CityScapes dataset provides depth information extracted from stereo vision, and annotations of 19 semantic classes (road, sidewalk, person, rider, car, truck, bus, on rails, motorcycle, bicycle, building, wall, fence, pole, traffic sign, traffic light, vegetation, terrain, and sky), in which 8 of them have instance-level annotations (person, rider, car, truck, bus, train, motorcycle and bicycle).

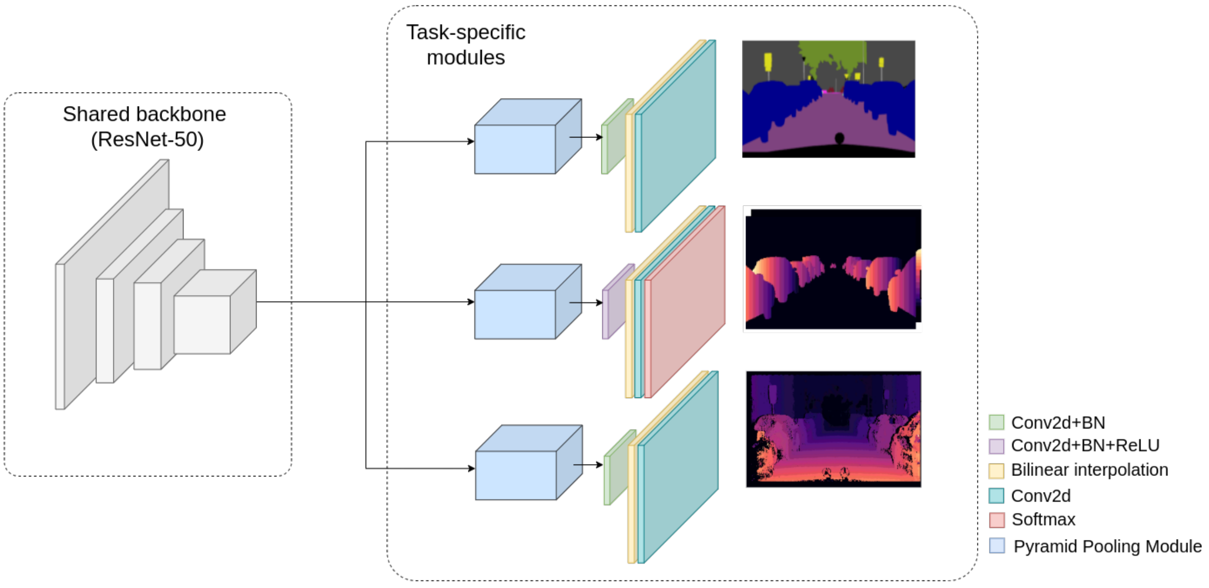
Since the test set annotations are not publicly available, the model evaluation was performed using the validation set (500 images). Hence, we randomly chose 275 images from the training set for validation and kept the remaining 2700 images for training. Moreover, due to computational limitations, the images and annotations were downsampled by a factor of four, reducing their original resolution from 1024×2048 to 256×512.

In addition to reducing the image's resolution, since the output space of the instance segmentation is unconstrained and we do not have information on how many object instances there are in an image, we defined the instance segmentation task as a regression problem based on a proposal-free approach, in which each instance is represented by a set of pixels that points to its center of mass (Figure 21) (WATANABE; WOLF, 2018; NAKAMURA; GRASSI; WOLF, 2021). Therefore, considering an object instance I, the displacement of each pixel with respect to its center of mass is calculated by:

$$\begin{aligned} d_x &= x - x_{cm}; \quad \{\forall_x \in I\} \\ d_y &= y - y_{cm}; \quad \{\forall_y \in I\} \end{aligned}$$

where (x_{cm}, y_{cm}) are the coordinates of each instance's center of mass (WATANABE; WOLF, 2018).

Figure 22 – Multi-task learning architecture for simultaneous learning of semantic segmentation, instance segmentation, and disparity estimation. See Section 5.4.2 for more details. Better visualization in color.



Source: Elaborated by the author.

Thus, the instances masks can be obtained by clustering the pixels that point to the same center (CIPOLLA; GAL; KENDALL, 2018; WATANABE; WOLF, 2018). However, note that this regression approach does not provide information about instance classes, making it necessary to have another model that provides the classes to estimate instance segmentation masks, such as classification (WATANABE; WOLF, 2018) or semantic segmentation (CIPOLLA; GAL; KENDALL, 2018; SENER; KOLTUN, 2018). In this work, we used the output of the semantic segmentation task to obtain regions of the image that contain the instance classes (NAKAMURA; GRASSI; WOLF, 2021).

Regarding the depth estimation task, since CityScapes provides disparity annotations rather than depth estimation, we trained the network to learn disparity estimation as in Cipolla, Gal and Kendall (2018), Neven *et al.* (2017), Sener and Koltun (2018). Furthermore, as in Sener and Koltun (2018), the disparity annotations were normalized according to the mean and standard deviation of the training set. Once we have the disparity maps, we can use the stereo camera's parameters given by CityScapes to compute the depth values.

5.4.2 Implementation Details

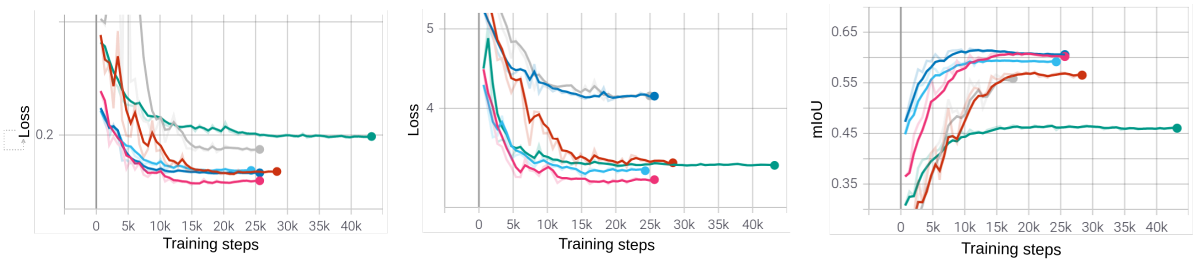
The architecture was adapted from Sener and Koltun (2018), in which the shared encoder consists of a ResNet-50 (HE *et al.*, 2016), and Pyramid Pooling Module (ZHAO *et al.*, 2017) is used to extract task-specific features. However, since the output size of the Pyramid Pooling Module is the same as its input, we need to retrieve the original input size of 256×512 pixels. Therefore, after the Pyramid Pooling Module, we apply a `1x1 conv` layer to reduce the feature maps' dimension, transforming the input feature channels to the target feature channels. In sequence, we apply a bilinear interpolation and add a `3x3 conv` to retrieve the original input resolution and refine the generation of the final prediction maps. For the classification task, we added a Rectified Linear Unit (ReLU) activation function before linear interpolation and a softmax layer at the end of the `3x3 conv` layer to get the probability of each object class (Figure 22). As in the handwritten digit experiments, we used the NLL loss for the classification task and the MSE loss for the regression tasks.

We used batch size 8 with random shuffle at every epoch, and we initialized the encoder parameters with a model pre-trained on ImageNet (DENG *et al.*, 2009). Each model was trained three times with a different seed value to get deterministic random data, and with the Adam optimizer (KINGMA; BA, 2015) with a learning rate of $1e-4$. Furthermore, the training process was interrupted only when there was no improvement in validation metrics during ten epochs.

5.4.3 Evaluation

Figure 23 shows the validation learning curves. Observing Figure 23c, the min-norm method (SENER; KOLTUN, 2018) (dark blue line) resulted in the fastest convergence for the semantic segmentation task. This is because this task has the lowest gradient magnitude among the other tasks. Consequently, the method could not adequately learn the instance segmentation due to its greater gradient magnitude, since it is underweighted during training (Figure 23b). The same happens with uniform weighting (green lines), the model obtains a good convergence for instance segmentation, but not for the others. Given that the central direction (KATRUTSA *et al.*, 2020) assigns the same importance to all tasks regardless of the magnitude of the gradient, this direction tends to find a good balance between tasks (red lines), as shown in previous experiments.

Figure 23 – Validation learning curves using different methods to simultaneously learn instance segmentation, semantic segmentation, and disparity estimation tasks. Gray lines represent when each model is trained separately for each task. Green ones represent the uniform weighting approach. Dark blue and red lines represent learning curves of min-norm Sener and Koltun (2018) and central dir. Katrutsa *et al.* (2020) methods, respectively. Finally, light blue and pink ones represent the learning curves using our proposed method to adapt the min-norm and central directions, respectively.



(a) Disparity estimation (b) Instance segmentation (c) Semantic segmentation

Source: Elaborated by the author.

However, when we add our tensioners to adapt the common descent directions, we can see that we increase the convergence speed and the learning curves become more consistent and smooth (light blue and pink lines).

Table 6 presents the evaluation results performed on the test set. For instance segmentation task, we adopt the same procedure as Sener and Koltun (2018) and Cipolla, Gal and Kendall (2018), i.e., evaluating the pixel direction estimation with respect to the center of mass of each instance. We report the Root Mean Squared Error (RMSE) for instance segmentation and disparity estimation tasks, and Mean Intersection over Union (mIoU) for semantic segmentation.

The first row shows the model performance when it is trained individually for each task. We define these performances as baseline to evaluate the methods. As it can be seen, when we define the same importance for all tasks, as adopted in most of the works that perform multi-task learning, this weighting improved the performance related to instance segmentation (second row). However, regarding the other tasks, the performance was worse than training single-task models. On the other hand, the method proposed by Sener and Koltun (2018) resulted in the best performance in semantic segmentation and the worst in instance segmentation (third row). This is because, in addition to the tasks having different descent directions, they also have variations in the order of magnitude. Consequently, this method underweighted the instance segmentation task so that it does not dominate the others during training. These behaviors can be observed in the analysis performed by Cipolla, Gal and Kendall (2018) and in the discussion of Figure 7 presented in Section 3.

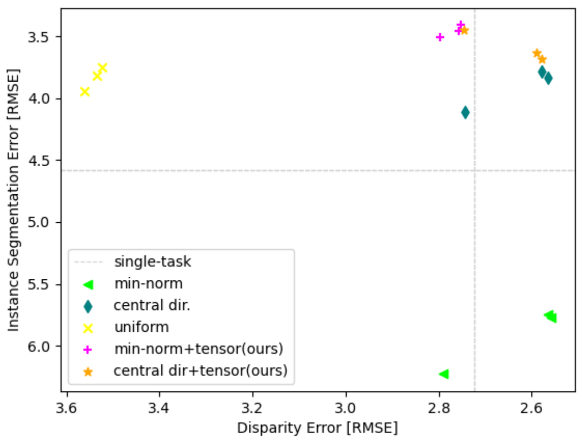
The third row presents the results with the central direction method, i.e. the direction that gives equal importance to all tasks, regardless of the gradient norm. We can see that it was possible to improve performance in almost all tasks compared to individual models.

Finally, the fifth and sixth rows present the results with the proposed method to change the minimum-norm and central directions. When comparing the minimum-norm direction (third and fifth rows), we can observe that the use of tensioners improved the performance in instance segmentation and kept the other two tasks with performance close to the single-task models.

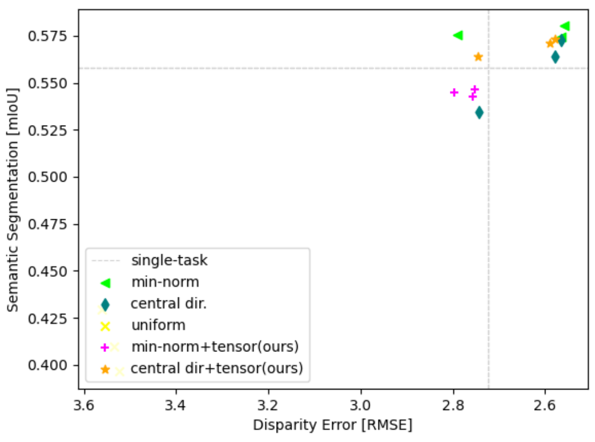
Table 6 – Average results of semantic segmentation, instance segmentation, and disparity estimation. We report mIoU for classification (higher is better) and RMSE for regression tasks (lower is better). Bold values represent the best results, while underlined ones represent the worst.

Descent direction method	Semantic segmentation (mIoU)$\uparrow$	Instance segmentation (RMSE)$\downarrow$	Disparity Error (RMSE)$\downarrow$
Single-task	0.5578 ± 0.0244	4.5766 ± 1.0583	2.7235 ± 0.2834
Uniform-weighting	$\underline{0.4119 \pm 0.0167}$	3.8348 ± 0.0986	$\underline{3.5404 \pm 0.0197}$
Min-norm	$\mathbf{0.5766 \pm 0.0031}$	$\underline{5.9152 \pm 0.2725}$	2.6381 ± 0.1324
Central dir.	0.5571 ± 0.0199	3.9077 ± 0.1784	$\mathbf{2.6289 \pm 0.0993}$
Min-norm + tensor(ours)	0.5447 ± 0.0017	$\mathbf{3.4581 \pm 0.0506}$	2.7686 ± 0.0237
Central dir. + tensor(ours)	0.5695 ± 0.0047	3.5871 ± 0.1265	2.6382 ± 0.0935

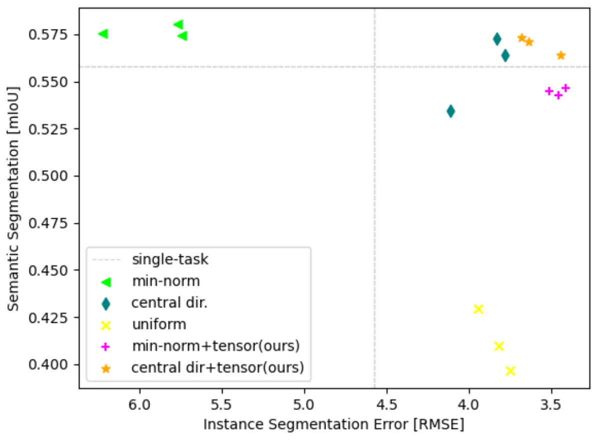
Figure 24 – Comparative plots of all trained models on the scene understanding problem. We use as baseline the average performance of single tasks (gray dashed line). The proposed method resulted in a good balance between tasks, resulting in a performance similar or even better than single-task models. Upper-right is better.



(a)



(b)



(c)

Source: Elaborated by the author.

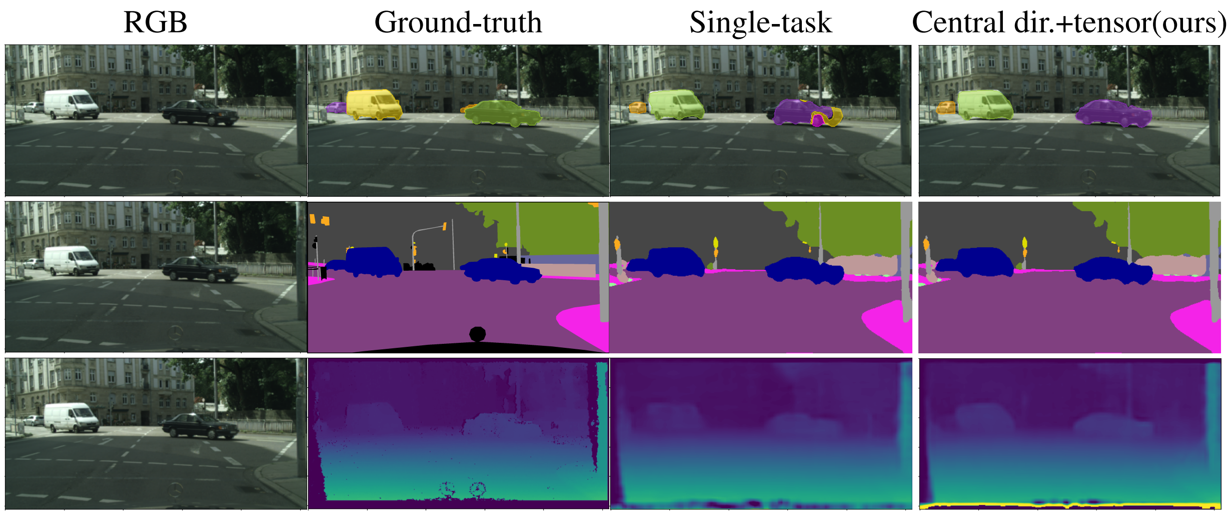
Regarding the central direction (fourth and sixth rows), we can observe that the use of tensioners resulted in a good balance between tasks, allowing us to overcome the performance of the baselines.

Figure 24 shows a comparative plot of all trained models on instance segmentation, semantic segmentation, and disparity estimation, in which the methods with the best results are those closest to the upper-right corner. As we can see, the use of tensioners generated consistent results in the three trained tasks, where in general it was possible to outperform the other evaluated methods. This highlights that there are many trade-offs between conflicting tasks, and the proposed method was able to find a balance that maximizes the performance of all of them.

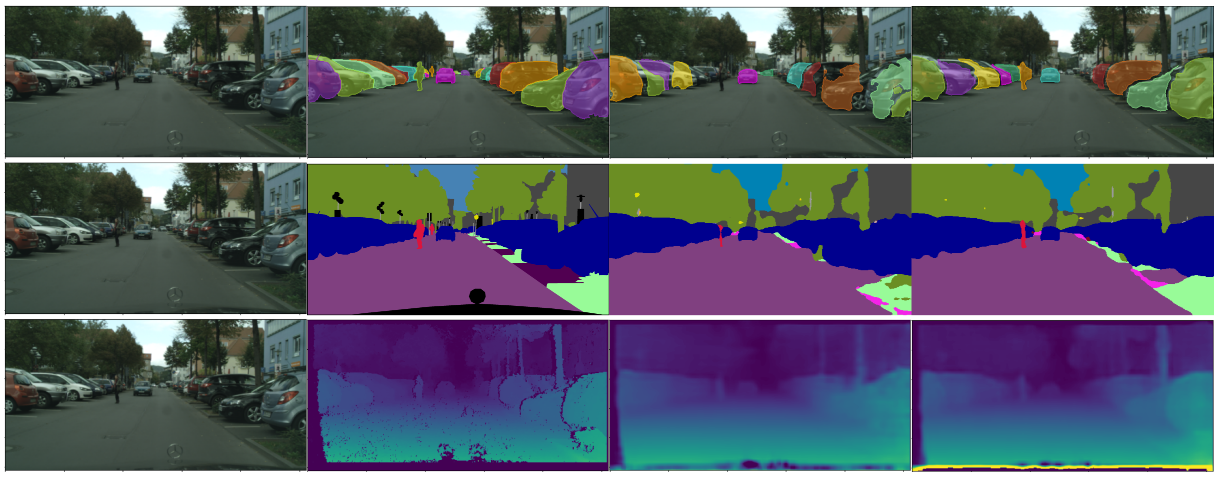
Some qualitative results of our proposed method are shown in Figures 25-26. We can observe that models trained separately generate inconsistent instance masks in cases of objects with illumination variations and reflections (third column of Figure 25). On the other hand, taking advantage of information from complementary tasks, such as the uniformity of instance and semantic segmentation masks, as well as the discontinuity of distance values between objects, allows our predictions to be uniform and with well-defined contours (fourth column of Figure 25). Moreover, since we trained each model in a downsampled image resolution of 256×512 to reduce the computational cost, it is important to mention that training the model in a lower resolution considerably affects the performance, especially in the case of smaller objects.

Regarding the disparity estimation, We can observe that the annotations consist of non-dense maps with several pixels with invalid values (second column of Figure 26). This is because the disparity annotations from CityScapes Dataset are computed using the Semi-global Matching (SGM). Therefore, when we train a model individually for disparity estimation, the result of the model ends up being affected by the non-dense maps (third column from Figure 26). On the other hand, the proposed method reduces the influence of noise caused by non-dense maps by properly taking advantage of information from related tasks, resulting in smoother disparity estimations and clearly defined object contours (fourth column from Figure 26).

Figure 25 – Qualitative comparative analysis of the proposed approach with single task models on CityScapes Dataset. Note that the results are grouped into blocks of three image sequences. For each block, the first row refers to the instance segmentation, the second to the semantic segmentation, and the third to the disparity estimation.



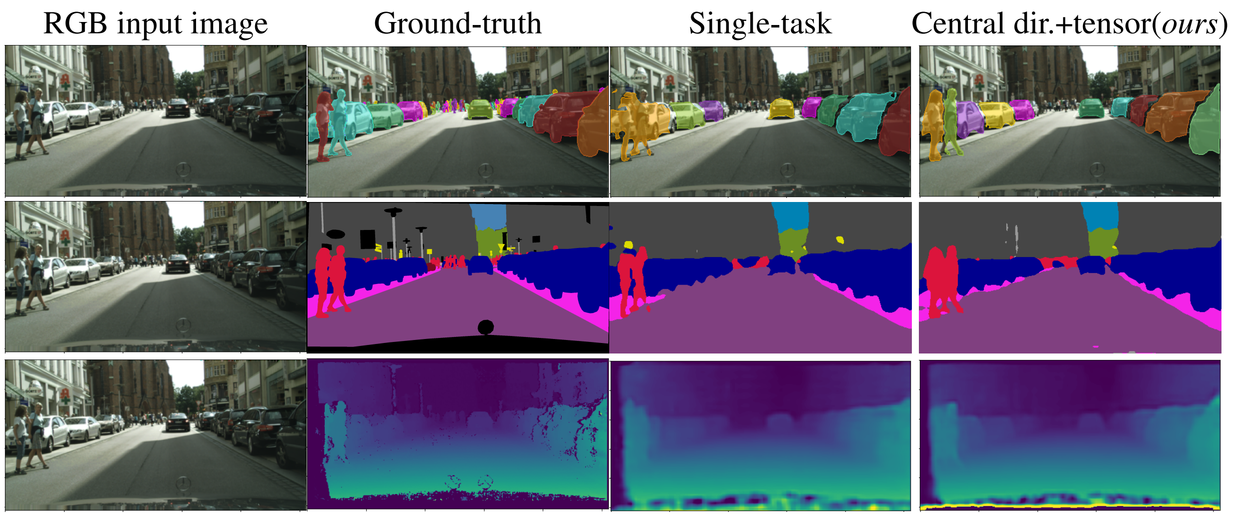
(a)



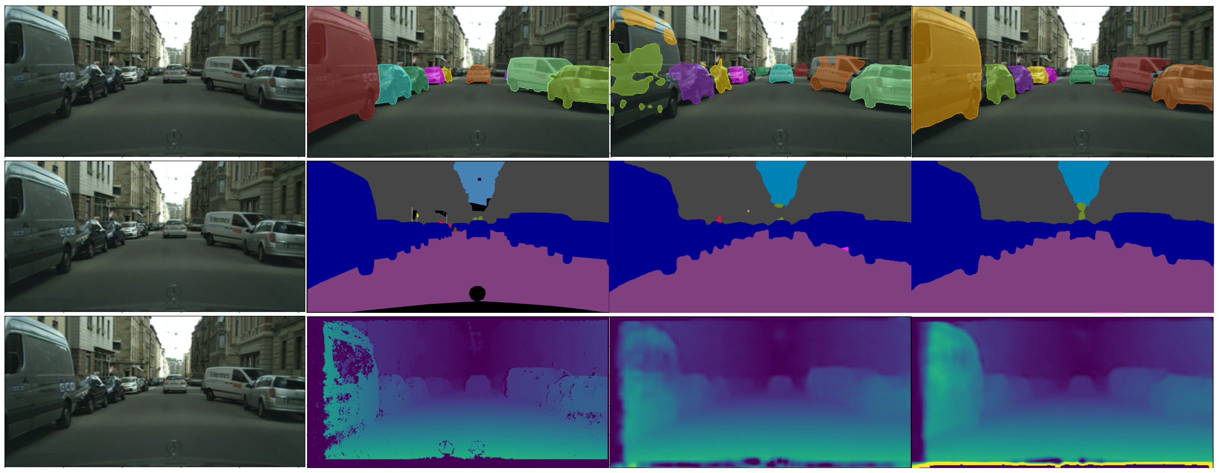
(b)

Source: Elaborated by the author.

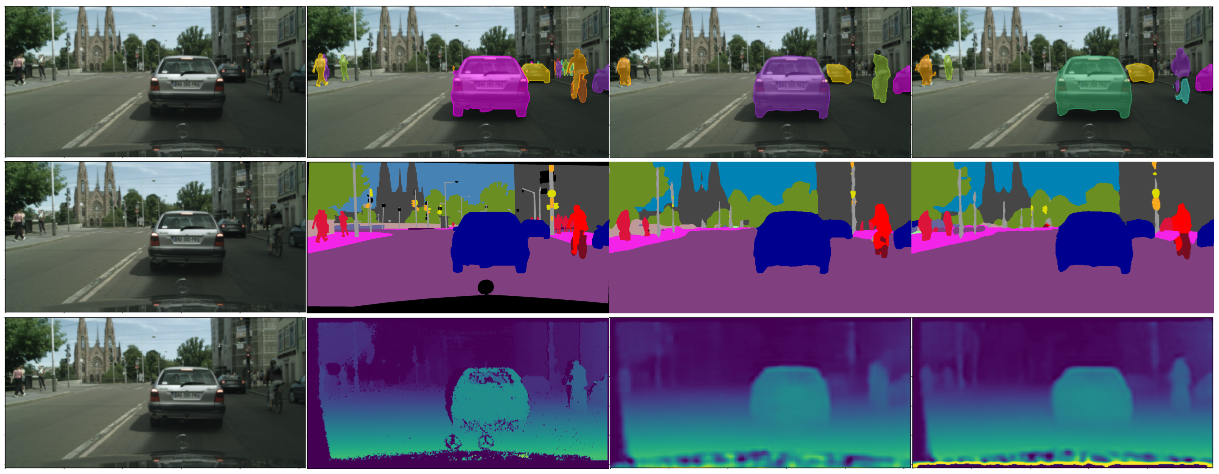
Figure 26 – Qualitative comparative analysis of the proposed approach with single task models on CityScapes Dataset. Note that the results are grouped into blocks of three image sequences. For each block, the first row refers to the instance segmentation, the second to the semantic segmentation, and the third to the disparity estimation.



(a)



(b)



(c)

Source: Elaborated by the author.

5.4.4 Memory Usage and Execution Time

Table 7 presents a comparative analysis of the number of trainable parameters in single-task and multi-task learning architectures. As it can be seen, it takes a total of 23.42 million parameters to learn the three tasks in a multi-task architecture. On the other hand, if individual architectures are used to learn each task, the number of trainable parameters increases to 45.8 million parameters, since the encoder is no longer shared between tasks. Regarding execution time and memory usage, it is positively correlated to the number of trainable parameters in the network architecture (BIANCO *et al.*, 2018). For example, the multi-task architecture adopted in this experiment performed the inferences at 78.35 fps. To compare with individual architectures, we performed the inferences of the three tasks serially, resulting in an inference time of 30.32 fps. To reduce inference time on individual architectures, one option would be to run them in parallel. However, it would result in an increase in memory usage. This difference becomes even more evident with the increase in the number of tasks or the more trainable parameters the network has. Therefore, since most of the model parameters in multi-task architectures are shared between tasks, it becomes computationally more efficient than single-task architectures.

Table 7 – Comparative analyses of the number of trainable parameters.

Method	**Number of parameters**			
	Semantic segmentation	**Instance segmentation**	**Disparity estimation**	**Total**
Single-tasks	15,286,099	15,246,914	15,244,609	45,777,622
Multi-task	23,424,598			23,424,598

5.5 Final Considerations

This chapter presents the experiments executed for validation of the proposed method. The results show that our method is robust to different applications, obtaining consistent results in different domains.

First, to isolate the method from other components that can influence the results, we performed sensitivity analysis on a simple multi-task problem of logical operations. In this study, we focused on the analysis of the descent directions behaviors between the proposed method and other existing methods. The results evidenced that the proposed method can deal with multiple tasks that present loss gradients with similar or different magnitudes. In both cases, the proposed method overcame the multi-task learning efficiency compared to other methods by speeding up the convergence curve and considerably improving the performance of all tasks together. This is because the existing methods have a static bias that tends to benefit only some tasks, especially when the tasks have different gradient magnitudes. For example, the minimum-norm method

(SENER; KOLTUN, 2018) tends to benefit the tasks with lower gradient magnitudes, while the proposed method dynamically adapts the common direction to give more importance to tasks that are diverging.

After the sensitivity analysis, we presented a second experiment on handwritten digit images. The goal was to validate the proposed method in a different and more complex application domain while keeping the tasks closely related and conflicting during training. In this multi-task problem, the input is given by an image containing two overlapping random digits. One of the tasks is to classify the upper left digit while the other is to reconstruct the lower right digit by an autoencoder. The obtained results demonstrate that the proposed method can maximize the performance of all tasks simultaneously without benefiting the performance of one task by degrading the other like in other methods.

Lastly, the third experiment addressed a recent and realistic vision-based problem, which was related to urban scene understanding. In this problem, the method had to learn image disparity estimation, instance segmentation, and semantic segmentation. Different from the previous experiments, the proposed method presented considerable gain in the final performance of all tasks only compared to the uniform weighting and the minimum-norm methods, while kept a very similar performance to the central direction. However, when observing the training process, the proposed method clearly sped up the convergence curves and made them more consistent and smooth.

CHAPTER 6 CONCLUSION

Multi-task learning (MTL) has gained attention due to its efficiency and potential to improve generalization performance. During the last years, several researchers around the world have been dedicated to improving the performance of MTL in various applications. These researches can be divided into two branches of work. One of them aims to develop more efficient architectures to reach better performances with less computation resources. The other concentrates on improving the optimization algorithms to ensure more consistent and faster learning convergence, which is the focus of this thesis.

Isolating the optimization problem from the architecture, the key point to be understood in multi-task learning is the fact that there are parameters to be optimized that will receive the influence of biases from multiple tasks. It means that those shared parameters will be updated by more than one gradient at once. Besides, the gradients can have different directions at some iterations. When it happens, the tasks are said to be conflicting. If the tasks are not conflicting, this implies that they have very similar features to be learned and their biases will be very likely the same. Consequently, the gradients of their loss functions will have very close directions. The optimization problem for MTL starts when the tasks to be learned simultaneously become conflicting with each other. In cases like this, the tasks will tend to compete for the same resource (e.g., neurons or convolutional kernels) by backpropagating gradients with different directions. If the optimization algorithm gives preference to the gradient of one task over the others, it will very probably improve the performance of the network on the benefited task in function of the degradation of the others. Therefore, the main objective of MTL algorithms is to find the appropriate combination of the gradients so that all tasks can converge simultaneously to an optimal solution.

We presented a broad review of related works, evidencing the state-of-the-art in MTL. In summary, existing methods focus on finding a policy to estimate the best linear combination of gradients in order to obtain a representative common descent direction between tasks. For instance, Sener and Koltun (2018) proposed to formulate the MTL as a Multi-objective Optimization (MOO) problem to find a common descent direction that minimizes the convex combination of the tasks' gradients. Also, later Katrutsa *et al.* (2020) proposed to give the same importance to all tasks by computing the descent direction that lies in the center of the cone formed by the gradient vectors.

Although those methods seem to be promising, there is a gap that avoids them to reach better results. This gap is the fact that they get only a snapshot of the gradients from the current training batch to compute the common descent direction, which makes them create a static bias without considering any indication from previous iterations that the computed directions are being representative for all tasks. Because of this, they cannot ensure that all tasks are converging to their optimal performances, although they are meeting the optimality conditions. From this observation, the hypothesis of this thesis is that a dynamic bias modeled by the convergence behavior of all tasks allows the computation of a more appropriate common descent direction that guarantees better convergence, without ignoring the performances of all tasks.

In this way, this work proposes a method that can adapt the common descent direction by giving more importance to the tasks that are diverging or underperforming on previous iterations. This is done by adopting the idea of tensioners that link the gradient of each task to the common descent direction vector, where the tensor strengths vary proportionally to the convergence of each task. To validate the hypothesis, experiments were conducted to compare the convergence curves between the proposed method and the existing state-of-the-art methods in a simple MTL problem, showing that the use of tensioners indeed can adapt the common descent direction to guarantee better convergence of all tasks, avoiding any task to be negatively impacted by a static bias. As consequence, it also significantly sped up the training process. Moreover, aiming to validate that the proposed method is applicable to more complex problems, two more experiments were performed on vision-based problems. In both of them, the method provided consistent performance with low variance, outperforming the state-of-the-art methods in simultaneous learning of conflicting tasks.

Despite the fact that the tensioners method can improve the simultaneous convergence of multiple tasks, one must understand that this is only possible if the behavior of the gradient variance of each task can be effectively abstracted. This work used the common moving average of gradients over the last T iterations as the abstraction function due to its generality. However, as a suggestion for future works, a deeper study of different functions would be necessary to understand how they affect the learning convergence of multiple tasks in different domains. For example, in cases where the convergence behavior of the tasks has less variance, the exponential moving average could be used instead of using a common moving average, since the exponential moving average is more sensitive to variations. In scenarios where convergence curves are less noisy, it tends to have fewer gradient outliers. Therefore, a function with more sensitivity to variance could be beneficial to accelerate the MTL with the tensioner method.

Another important research front that could be explored after this thesis is the combination of the proposed optimization method with more elaborated network architectures. As presented in Section 3.1, there are plenty of works proposing different network configurations to allow more efficient allocation of task-specific and common feature layers, instead of using a simple tree-like architecture. According to (VANDENHENDE *et al.*, 2022), to design an appropriate network architecture for MTL, one must know the set of tasks to be solved and their relationship. Once it is done and the network configuration is set to optimize the usage of parameters, the tensioners method could be used to optimize the convergence of shared parameters during the training process.

Lastly, since our method is independent of a specific task, it is possible to extend our approach to other applications, such as robotics or intelligent vehicles, where several related tasks must be estimated simultaneously with reduced inference time and computational resource usage. Also, the proposed method could be applied to learn tasks with supervised, unsupervised, and/or reinforcement learning, as the framework proposed by Horita *et al.* (2021) that allows learning autonomous navigation policy through reinforcement learning at the same time it learns semantic segmentation of urban scenario through supervised learning. In addition to multi-task learning, we believe that it would also be possible to use the tensioners method to explore other areas where it is necessary to find a trade-off between learning multiple tasks, such as dealing with the problem of catastrophic forgetting in continuous learning, restricting the learning of a new task to be according to a descent direction that is representative for the previously learned tasks.

BIBLIOGRAPHY

BAYOUDH, K.; KNANI, R.; HAMDAOUI, ·. F.; Abdellatif Mtibaa, ·.; Khaled Bayoudh, B.; HAMDAOUI, F.; MTIBAA, A. A survey on deep multimodal learning for computer vision: advances, trends, applications, and datasets. **The Visual Computer 2021**, Springer, p. 1–32, jun 2021. ISSN 1432-2315. Available: <https://link.springer.com/article/10.1007/s00371-021-02166-7>. Citations on pages 39 and 40.

BIANCO, S.; CADENE, R.; CELONA, L.; NAPOLETANO, P. Benchmark analysis of representative deep neural network architectures. **IEEE Access**, v. 6, p. 64270–64277, 2018. Citation on page 73.

BOTTOU, L.; CURTIS, F. E.; NOCEDAL, J. Optimization Methods for Large-Scale Machine Learning. 2018. Available: <https://arxiv.org/abs/1606.04838v3>. Citation on page 38.

CARUANA, R. **Multitask Learning**. 255 p. Phd Thesis (PhD Thesis) — Carnegie Mellon University, 1997. Citations on pages 27 and 32.

CARUANA, R. A. Multitask Learning: A Knowledge-Based Source of Inductive Bias. In: **Machine Learning Proceedings 1993**. [S.l.: s.n.], 1993. p. 41–48. Citations on pages 32, 33, and 40.

CHEN, S.; ZHANG, Y.; YANG, Q. Multi-Task Learning in Natural Language Processing: An Overview. sep 2021. Available: <https://arxiv.org/abs/2109.09138v1>. Citations on pages 39, 40, and 41.

CHEN, Z.; BADRINARAYANAN, V.; LEE, C. Y.; RABINOVICH, A. GradNorm: Gradient normalization for adaptive loss balancing in deep multitask networks. **35th International Conference on Machine Learning, ICML 2018**, p. 1240–1251, nov 2018. Citations on pages 27 and 41.

CHENNUPATI, S.; SISTU, G.; YOGAMANI, S.; RAWASHDEH, S. AuxNet: Auxiliary Tasks Enhanced Semantic Segmentation for Automated Driving. In: **Proceedings of the 14th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications**. [S.l.: s.n.], 2019. v. 5, p. 645–652. Citations on pages 28 and 41.

CIPOLLA, R.; GAL, Y.; KENDALL, A. Multi-task Learning Using Uncertainty to Weigh Losses for Scene Geometry and Semantics. In: **2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition**. [S.l.]: IEEE, 2018. abs/1705.0, p. 7482–7491. Citations on pages 15, 27, 41, 42, 66, and 68.

CORDTS, M.; OMRAN, M.; RAMOS, S.; REHFELD, T.; ENZWEILER, M.; BENENSON, R.; FRANKE, U.; ROTH, S.; SCHIELE, B. The Cityscapes Dataset for Semantic Urban Scene Understanding. In: **2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)**. [S.l.]: IEEE, 2016. p. 3213–3223. ISBN 978-1-4673-8851-1. ISSN 10636919. Citation on page 65.

CRAWSHAW, M. Multi-Task Learning with Deep Neural Networks: A Survey. sep 2020. Available: <https://arxiv.org/abs/2009.09796v1>. Citations on pages 39 and 42.

DENG, J.; DONG, W.; SOCHER, R.; LI, L.-J.; Kai Li; Li Fei-Fei. ImageNet: A large-scale hierarchical image database. **2009 IEEE Conference on Computer Vision and Pattern Recognition**, IEEE, p. 248–255, 2009. Citation on page 67.

DÉSIDÉRI, J. A. Multiple-gradient descent algorithm (MGDA) for multiobjective optimization. **Comptes Rendus Mathematique**, Elsevier Masson SAS, v. 350, n. 5-6, p. 313–318, 2012. ISSN 1631073X. Citations on pages 38 and 49.

DWIVEDI, K.; ROIG, G. Representation similarity analysis for efficient task taxonomy transfer learning. In: **Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)**. [S.l.: s.n.], 2019. Citation on page 41.

EHRGOTT, M. **Multicriteria optimization**. 2 ed. ed. Berlin/Heidelberg: Springer-Verlag, 2005. 1–323 p. ISBN 3540213988. Citations on pages 28, 34, 36, and 43.

EIGEN, D.; FERGUS, R. Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. In: **Proceedings of the IEEE International Conference on Computer Vision**. [S.l.: s.n.], 2015. p. 2650–2658. ISBN 9781467383912. ISSN 15505499. Citation on page 28.

FLIEGE, J.; SVAITER, B. F. Steepest descent methods for multicriteria optimization. **Mathematical Methods of Operations Research**, v. 51, n. 3, p. 479–494, 2000. ISSN 14322994. Citation on page 36.

FUKUDA, E. H.; DRUMMOND, L. M. G. A SURVEY ON MULTIOBJECTIVE DESCENT METHODS. **Pesquisa Operacional**, v. 34, n. 3, p. 585–620, dec 2014. ISSN 0101-7438. Citation on page 36.

GUNANTARA, N. A review of multi-objective optimization: Methods and its applications. **Cogent Engineering**, Cogent, v. 5, n. 1, p. 1–16, jan 2018. ISSN 23311916. Available: <https://www.tandfonline.com/doi/abs/10.1080/23311916.2018.1502242>. Citation on page 41.

GUO, M.; HAQUE, A.; HUANG, D. A.; YEUNG, S.; FEI-FEI, L. Dynamic Task Prioritization for Multitask Learning. In: **Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)**. [S.l.: s.n.], 2018. p. 282–299. ISSN 16113349. Citations on pages 39 and 41.

HAI, Z.; ZHAO, P.; CHENG, P.; YANG, P.; LI, X.-L.; LI, G. Deceptive review spam detection via exploiting task relatedness and unlabeled data. In: **Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing**. Austin, Texas: Association for Computational Linguistics, 2016. p. 1817–1826. Available: <https://aclanthology.org/D16-1187>. Citation on page 39.

HARADA, K.; SAKUMA, J.; KOBAYASHI, S. Local search for multiobjective function optimization: Pareto descent method. In: **GECCO 2006 - Genetic and Evolutionary Computation Conference**. [S.l.: s.n.], 2006. v. 1, p. 659–666. ISBN 1595931864. Citations on pages 37 and 43.

HAYKIN, S. **Neural Networks and Learning Machines**. Prentice Hall, 2009. (Neural networks and learning machines, v. 10). ISBN 9780131471399. Available: <https://books.google.com.br/books?id=K7P36lKzI_QC>. Citation on page 47.

HE, K.; ZHANG, X.; REN, S.; SUN, J. Deep residual learning for image recognition. **Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition**, p. 770–778, dec 2016. Citation on page 66.

HORITA, L. R. T.; NAKAMURA, A. T. M.; WOLF, D. F.; JUNIOR, V. G. Improving multi-goal and target-driven reinforcement learning with supervised auxiliary task. In: **2021 20th International Conference on Advanced Robotics (ICAR)**. [S.l.: s.n.], 2021. p. 290–295. Citation on page 77.

HWANG, C.-L.; MASUD, A. S. M. **Multiple Objective Decision Making — Methods and Applications**. Berlin, Heidelberg: Springer Berlin Heidelberg, 1979. (Lecture Notes in Economics and Mathematical Systems, v. 164). ISBN 978-3-540-09111-0. Available: <http://link.springer.com/10.1007/978-3-642-45511-7>. Citations on pages 34, 36, and 43.

KATRUTSA, A.; MERKULOV, D.; TURSYNBEK, N.; OSELEDETS, I. **Follow the bisector: a simple method for multi-objective optimization**. 2020. Citations on pages 17, 28, 43, 50, 54, 56, 61, 67, and 76.

KINGMA, D. P.; BA, J. L. Adam: A method for stochastic optimization. In: **3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings**. [S.l.: s.n.], 2015. Citation on page 67.

KOKKINOS, I. Ubernet: Training a universal convolutional neural network for low-, mid-, and high-level vision using diverse datasets and limited memory. In: . [S.l.]: Institute of Electrical and Electronics Engineers Inc., 2017. v. 2017-January, p. 5454–5463. ISBN 9781538604571. Citations on pages 40 and 41.

LECUN, Y.; BOSER, B.; DENKER, J. S.; HENDERSON, D.; HOWARD, R. E.; HUBBARD, W.; JACKEL, L. D. Backpropagation Applied to Handwritten Zip Code Recognition. **Neural Computation**, v. 1, n. 4, p. 541–551, dec 1989. ISSN 0899-7667. Citation on page 32.

LECUN, Y.; BOTTOU, L.; BENGIO, Y.; HAFFNER, P. Gradient-based learning applied to document recognition. **Proceedings of the IEEE**, v. 86, n. 11, p. 2278–2323, 1998. ISSN 00189219. Citation on page 62.

LI, C.; YAN, J.; WEI, F.; DONG, W.; LIU, Q.; ZHA, H. Self-Paced Multi-Task Learning. **31st AAAI Conference on Artificial Intelligence, AAAI 2017**, p. 2175–2181, apr 2016. Citation on page 41.

LI, J.; LIU, X.; YIN, W.; YANG, M.; MA, L.; JIN, Y. Empirical evaluation of multi-task learning in deep neural networks for natural language processing. **Neural Computing and Applications**, Springer Science and Business Media Deutschland GmbH, v. 33, n. 9, p. 4417–4428, may 2021. ISSN 14333058. Available: <https://link.springer.com/article/10.1007/s00521-020-05268-w>. Citation on page 41.

LIN, X.; ZHEN, H.-L.; LI, Z.; ZHANG, Q.-F.; KWONG, S. Pareto multi-task learning. In: WALLACH, H.; LAROCHELLE, H.; BEYGELZIMER, A.; ALCHé-BUC, F. d'; FOX, E.; GARNETT, R. (Ed.). **Advances in Neural Information Processing Systems**. [S.l.]: Curran Associates, Inc., 2019. v. 32. Citations on pages 42 and 43.

LIU, P.; QIU, X.; HUANG, X. Adversarial multi-task learning for text classification. In: **ACL 2017 - 55th Annual Meeting of the Association for Computational Linguistics, Proceedings**

of the Conference (Long Papers). [S.l.]: Association for Computational Linguistics, 2017. v. 1, p. 1–10. ISBN 9781945626753. Citation on page 40.

LIU, P.; QIU, X.; XUANJING, H. Recurrent neural network for text classification with multi-task learning. In: **IJCAI International Joint Conference on Artificial Intelligence**. [S.l.: s.n.], 2016. v. 2016-Janua, p. 2873–2879. ISSN 10450823. Citation on page 27.

LIU, S.; JOHNS, E.; DAVISON, A. J. End-To-End Multi-Task Learning With Attention. **2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)**, IEEE, p. 1871–1880, jun 2019. Citation on page 41.

MAHAPATRA, D.; RAJAN, V. Multi-Task learning with user preferences: Gradient descent with controlled ascent in pareto optimization. In: **37th International Conference on Machine Learning, ICML 2020**. [S.l.: s.n.], 2020. PartF16814, p. 6553–6563. ISBN 9781713821120. Citation on page 43.

MAO, Y.; YUN, S.; LIU, W.; DU, B. Tchebycheff Procedure for Multi-task Text Classification. In: **Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics**. Stroudsburg, PA, USA: Association for Computational Linguistics, 2020. p. 4217–4226. Citations on pages 27 and 43.

MELLO, R.; PONTI, M. **Machine Learning: A Practical Approach on the Statistical Learning Theory**. Springer International Publishing, 2018. ISBN 9783319949888. Available: <https://books.google.com.br/books?id=N3ectwEACAAJ>. Citation on page 47.

MIETTINEN, K. **Nonlinear Multiobjective Optimization**. [S.l.]: Springer US, 1998. 298 p. ISBN 9781461555636. Citations on pages 28, 34, 35, 36, 37, and 43.

MISRA, I.; SHRIVASTAVA, A.; GUPTA, A.; HEBERT, M. Cross-Stitch Networks for Multi-task Learning. In: **Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition**. [S.l.: s.n.], 2016. v. 2016-Decem, p. 3994–4003. Citations on pages 39 and 41.

NAKAMURA, A. T. M.; GRASSI, V.; WOLF, D. F. An effective combination of loss gradients for multi-task learning applied on instance segmentation and depth estimation. **Engineering Applications of Artificial Intelligence**, Pergamon, v. 100, p. 104205, apr 2021. ISSN 09521976. Citations on pages 27, 32, 42, 65, and 66.

NEVEN, D.; BRABANDERE, B. de; GEORGOULIS, S.; PROESMANS, M.; GOOL, L. van. **Fast scene understanding for autonomous driving**. 2017. Citation on page 66.

PLAZA-DEL-ARCO, F. M.; MOLINA-GONZALEZ, M. D.; URENA-LOPEZ, L. A.; MARTIN-VALDIVIA, M. T. A multi-task learning approach to hate speech detection leveraging sentiment analysis. **IEEE Access**, Institute of Electrical and Electronics Engineers Inc., v. 9, p. 112478–112489, 2021. ISSN 21693536. Citation on page 40.

REN, P.; XIAO, Y.; CHANG, X.; HUANG, P. Y.; LI, Z.; CHEN, X.; WANG, X. A Comprehensive Survey of Neural Architecture Search. **ACM Computing Surveys (CSUR)**, ACM PUB27 New York, NY, USA, v. 54, n. 4, p. 76, may 2021. ISSN 15577341. Available: <https://dl.acm.org/doi/10.1145/3447582>. Citations on pages 39 and 41.

ROSERO, L. A.; GOMES, I. P.; SILVA, J. A. R.; SANTOS, T. C. dos; NAKAMURA, A. T. M.; AMARO, J.; WOLF, D. F.; OSÓRIO, F. S. A software architecture for autonomous vehicles: Team lrm-b entry in the first carla autonomous driving challenge. **ArXiv**, abs/2010.12598, 2020. Citation on page 31.

RUDER, S.; BINGEL, J.; AUGENSTEIN, I.; SØGAARD, A. Latent Multi-Task Architecture Learning. **Proceedings of the AAAI Conference on Artificial Intelligence**, AAAI Press, v. 33, n. 01, p. 4822–4829, jul 2019. ISSN 2374-3468. Available: <https://ojs.aaai.org/index.php/AAAI/article/view/4410>. Citation on page 39.

SABOUR, S.; FROSST, N.; HINTON, G. E. Dynamic routing between capsules. In: **Advances in Neural Information Processing Systems**. [S.l.: s.n.], 2017. v. 2017-Decem, p. 3857–3867. ISSN 10495258. Citation on page 62.

SAMANT, R. M.; BACHUTE, M. R.; GITE, S.; KOTECHA, K. Framework for deep learning-based language models using multi-task learning in natural language understanding: A systematic literature review and future directions. **IEEE Access**, v. 10, p. 17078–17097, 2022. Citation on page 39.

SANCHEZ, D.; OLIU, M.; MADADI, M.; BARO, X.; ESCALERA, S. Multi-task human analysis in still images: 2D/3D pose, depth map, and multi-part segmentation. In: **2019 14th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2019)**. [S.l.]: IEEE, 2019. p. 1–8. ISBN 978-1-7281-0089-0. Citation on page 41.

SCHÄFFLER, S.; SCHULTZ, R.; WEINZIERL, K. Stochastic method for the solution of unconstrained vector optimization problems. **Journal of Optimization Theory and Applications**, v. 114, n. 1, p. 209–222, 2002. ISSN 00223239. Citation on page 38.

SENER, O.; KOLTUN, V. Multi-task learning as multi-objective optimization. In: **Advances in Neural Information Processing Systems**. [S.l.: s.n.], 2018. v. 2018-Decem, p. 527–538. ISSN 10495258. Citations on pages 17, 27, 28, 42, 49, 56, 59, 61, 62, 66, 67, 68, 74, and 75.

SERMANET, P.; EIGEN, D.; ZHANG, X.; MATHIEU, M.; FERGUS, R.; LECUN, Y. **OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks**. 2013. Citation on page 41.

TEICHMANN, M.; WEBER, M.; ZÖLLNER, M.; CIPOLLA, R.; URTASUN, R. MultiNet: Real-time Joint Semantic Reasoning for Autonomous Driving. In: **IEEE Intelligent Vehicles Symposium, Proceedings**. [S.l.: s.n.], 2018. p. 1013–1020. Citations on pages 28 and 41.

VANDENHENDE, S.; GEORGOULIS, S.; De Brabandere, B.; Van Gool, L. Branched Multi-Task Networks: Deciding What Layers To Share. apr 2019. Citations on pages 40 and 41.

VANDENHENDE, S.; GEORGOULIS, S.; Van Gansbeke, W.; PROESMANS, M.; DAI, D.; Van Gool, L. Multi-Task Learning for Dense Prediction Tasks: A Survey. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, IEEE Computer Society, v. 44, n. 7, p. 3614–3633, jul 2022. ISSN 19393539. Citations on pages 39, 41, 43, and 77.

WATANABE, T.; WOLF, D. Distance to Center of Mass Encoding for Instance Segmentation. In: **IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC**. [S.l.: s.n.], 2018. p. 3825–3831. ISBN 9781728103235. Citations on pages 65 and 66.

XIAO, L.; ZHANG, H.; CHEN, W. Gated multi-task network for text classification. In: **NAACL HLT 2018 - 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies - Proceedings of the Conference**. [S.l.: s.n.], 2018. v. 2, p. 726–731. ISBN 9781948087292. Citation on page 27.

ZHANG, Y.; YANG, Q. A Survey on Multi-Task Learning. **IEEE Transactions on Knowledge and Data Engineering**, p. 1–1, 2021. ISSN 1041-4347. Citations on pages 32 and 39.

ZHANG, Z.; LUO, P.; LOY, C. C.; TANG, X. Facial landmark detection by deep multi-task learning. **Lecture Notes in Computer Science**, Springer Verlag, v. 8694 LNCS, n. PART 6, p. 94–108, 2014. ISSN 16113349. Available: <https://link.springer.com/chapter/10.1007/978-3-319-10599-4_7>. Citation on page 40.

ZHAO, H.; SHI, J.; QI, X.; WANG, X.; JIA, J. Pyramid scene parsing network. **Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017**, p. 6230–6239, 2017. Citation on page 66.

GLOSSARY

mIoU Mean Intersection over Union.

ML Machine Learning.

MLP Multi-layer Perceptron.

MOO Multi-objective Optimization.

MSE Mean Squared Error.

MTL Multi-task Learning.

NLL Negative Log-likelihood.

ReLU Rectified Linear Unit.

RMSE Root Mean Squared Error.

SGD Stochastic Gradient Descent.

SGM Semi-global Matching.

TCDCN Tasks-Constrained Deep Convolutional Network.

APPENDIX

A

PUBLICATIONS

Published journal articles:

1. NAKAMURA, A. T. M.; WOLF, D. F.; GRASSI Jr., V. Leveraging convergence behavior to balance conflicting tasks in multi-task learning. In: Neurocomputing, v. 511, p. 43-53, 2022. ISSN 0925-2312. (A1)
2. NAKAMURA, A. T. M.; WOLF, D. F.; GRASSI Jr., V. An Effective Combination of Loss Gradients for Multi-Task Learning Applied on Instance Segmentation and Depth Estimation. In: Engineering Applications of Artificial Intelligence, v. 100, p. 104205, 2021. ISSN 0952-1976. (A1)

Published conference papers:

1. HORITA, L. R. T.; NAKAMURA, A. T. M.; WOLF, D. F.; GRASSI Jr., V. Improving Multi-goal and Target-driven Reinforcement Learning with Supervised Auxiliary Task. 2021 20th International Conference on Advanced Robotics (ICAR), 2021, pp. 290-295. (A3)

ICMC USP
SÃO CARLOS