Neurocomputing 511 (2022) 43–53

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

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

Angelica Tiemi Mizuno Nakamura [a,*], Valdir Grassi Jr [b], Denis Fernando Wolf [a]

[a] Mobile Robotics Laboratory, Institute of Mathematics and Computer Sciences, University of São Paulo, São Carlos, Brazil
[b] Department of Electrical and Computer Engineering, São Carlos School of Engineering, University of São Paulo, São Carlos, Brazil

ARTICLE INFO

Article history:
Received 15 December 2021
Revised 25 June 2022
Accepted 4 September 2022
Available online 9 September 2022

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

ABSTRACT

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. However, tasks often conflict with each other, which makes it challenging to define how the gradients of multiple tasks should be combined to allow simultaneous learning. To address this problem, we use the idea of multi-objective optimization to propose a method that takes into account temporal behaviour 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. As a result, we empirically show that the proposed method outperforms the state-of-the-art approaches on learning conflicting tasks. Unlike the adopted baselines, our method ensures that all tasks reach good generalization performances.

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 ([1]). Due to its potential to improve generalization performance, it has gained attention in several areas of scientific and industrial communities, such as computer vision ([2–5]) and natural language processing ([6–8]).

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 creates the 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 to implement 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 each task's gradients over the shared neurons ([9,3]). Therefore, the tasks will share and compete for the same resource (e.g., neurons or convolutional kernels), making it necessary to handle conflicting tasks. We define two tasks as conflicting when their gradients have different directions. Thus, when the descent direction of a task is not representative for

* Corresponding author.
E-mail addresses: tiemi.mizuno@usp.br (A.T.M. Nakamura), vgrassi@usp.br (V. Grassi Jr), denis@icmc.usp.br (D.F. Wolf).

https://doi.org/10.1016/j.neucom.2022.09.042
0925-2312/

the other, optimizing the network parameters in relation to a bad direction can harm 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, [2] 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 ([10,11]). 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 Section 3.

Based on the above considerations, this work proposes 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 ([2]) or the central direction between gradients ([12]). 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, 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. For this, the proposed method creates the idea of tensors 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 paper are as follows: 1) A method to create a dynamic learning bias by leveraging the convergence behavior of tasks. 2) An approach independent of a specific application to combine multiple gradient vectors and find a common feasible descent direction, aiming to maximize the performance of all tasks. To confirm the performance of the proposed method in learning conflicting tasks, we performed ablation studies and a series of experiments on a public handwritten digit classification dataset.

The remainder of this paper is structured as follows. Section 2 provides a literature review. In Section 3, we present our proposed method to combine gradients of multiple tasks by leveraging their convergence behavior over the past iterations. In Section 4, we perform ablation studies and compare our proposed method with other weighting methods, followed by a discussion of the results. Finally, we conclude in Section 5.

2. Related work

Introduced by [13], the Multi-Task Learning (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 may cause confusion among researchers. For example, one could say that MTL applied to neural networks is simply the fact of creating an architecture with multiple outputs, which has already been extensively explored. However, this kind of solution usually encodes a single task ([14,1,15]). On the other hand, the MTL aims to explore the idea that related tasks can create an inductive bias, and based on that bias, a certain task will prefer a hypothesis x out of the n possible hypothesis. In deep neural networks, multi-task learning can be categorized according to how parameters are shared between tasks, i.e., soft or hard parameter sharing.

In soft parameter sharing, each task has its own model with its own parameters, and the bias induction is accomplished through functions that reduce the distance between the models' parameters ([16–19]). However, this approach results in very complex architectures with higher computational costs, making it unfeasible for many applications. On the other hand, the hard parameter sharing approach allows a set of parameters to be shared among all tasks, reducing inference latency and storage costs ([13]). For example, in computer vision, the architecture of a hard parameter sharing model is usually composed of a shared encoder and multiple task-specific decoders, in which the models are trained by optimizing the gradient combination of all tasks. This is commonly done by a linear combination of the negative gradients of each task ([9,20–22]), or some variation with adaptive weights ([3,4,23–25]).

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 and compete for the same resources (e.g., convolutional kernels from encoder), this combination must be done by finding a trade-off between them so that shared layers can learn relevant features to all tasks ([3,5]). However, the linear combination does not allow finding this balance between tasks ([2,26]). As an alternative, [2] proposed to model multi-task learning as a multi-objective optimization problem to find a solution that is not dominated by any other (Pareto optimal solutions), achieving better results than previous linear combination methods and its variations. However, in problems with variations in the gradients' scale, these variations can significantly impact the final solution. In [2], his solution may prioritize only the tasks with lower gradient norm and not learning the other tasks well. To find a solution that considers the tasks equally important, [27,12] removed the scale influence by normalizing the gradients for each task. In [6], it is proposed to use the Tchebycheff metric to handle non-convex problems. Instead of finding a single optimal solution, [26,28] propose to find a set of Pareto optimal solutions with different trade-offs between tasks. However, these methods consider 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. It creates a tendency for the solution to prefer a specific direction. Therefore, this work proposes to analyze the convergence of tasks during the training process to dynamically correct the common descent direction and find a trade-off between conflicting tasks that maximize simultaneous learning. Thus, we 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. Through experimental results, we show that the proposed method can handle conflicting tasks, finding a good balance between them during the training process.

3. Approach

The idea of Multi-task Learning (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 ([13]). To

perform simultaneous learning of multiple tasks, we will consider a hard parameter sharing approach, in which a subset of hidden layers parameters with low-level features are shared between all tasks, while the rest of the high-level parameters are kept as task-specific (Fig. 1). Hence, considering a multi-task learning problem over a space X and a set of tasks $\{Y_m\}_{m\in[M]}$, the decision function of each task m can be defined as $f_m(x;\theta_{sh};\theta_m): X \to Y$. Where the parameters θ_{sh} are shared among tasks and θ_m are task-specific parameters.

Given a set of objective functions $\mathscr{L}(f_m(x;\theta_{sh},\theta_m))$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:

$$\min_{\substack{\theta_{sh},\\ \theta_1,\ldots,\theta_M}} \sum_{m=1}^{M} \frac{1}{N} \sum_{i=1}^{N} k_m \mathscr{L}(f_m(x_i;\theta_{sh},\theta_m), y_{i,m}),$$

where $y_{i,m}$ is the label of the m^{th} task of the i^{th} data, N is the number of data, and $k_m > 0$ are the predefined or dynamically calculated weighting coefficients associated with each objective function.

Although the linear combination is a simple method to solve multi-task learning problems, this approach cannot adequately model competing tasks ([2,28]). On the other hand, multi-objective optimization is an optimization problem that deals with multiple conflicting objective functions, making it an intuitive way to deal with problems where there is no single optimal solution for all objective functions ([10,11]).

3.1. Feasible descent direction

According to [29], a direction $\vec{v}$ can simultaneously improve all tasks if:

$$-\nabla_{sh,m} \cdot \vec{v} \geqslant 0, \forall m \in 1,\ldots,M, \quad (1)$$

where $\nabla_{sh,m}$ represents the gradient vector of the m^{th} task w.r.t to the shared parameters, sh. Thus, given the negative gradient vectors of three tasks, there will be several feasible descent directions. The set of all feasible descent directions is shown in gray in Fig. 2.

However, not all feasible descent directions will be representative for all objective functions. For example, considering the feasible descent directions $\vec{d}_1, \vec{d}_2, \vec{d}_3$ and $\vec{d}_4$, we can see that $\vec{d}_2$ decreases more each objective function than $\vec{d}_1$:

$$\begin{aligned}
\vec{d}_1 \cdot -\nabla_{sh,1}(\theta) &< \vec{d}_2 \cdot -\nabla_{sh,1}(\theta),\\
\vec{d}_1 \cdot -\nabla_{sh,2}(\theta) &< \vec{d}_2 \cdot -\nabla_{sh,2}(\theta),\\
\vec{d}_1 \cdot -\nabla_{sh,3}(\theta) &< \vec{d}_2 \cdot -\nabla_{sh,3}(\theta).
\end{aligned}$$

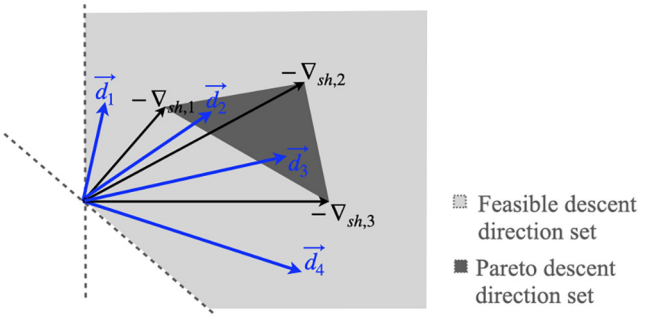


Fig. 2. Feasible set and Pareto descent directions considering the negative gradient vectors of $\nabla_{sh,1}, \nabla_{sh,2}$ and $\nabla_{sh,3}$. The blue vectors are examples of feasible descent directions given the set of negative gradients of the tasks. Best visualized with color.

Similarly, the same holds true for $\vec{d}_3$ and $\vec{d}_4$. On the other hand, $\vec{d}_2$ and $\vec{d}_3$ descent directions cannot improve an objective function without degrading the improvement of another:

$$\begin{aligned}
\vec{d}_2 \cdot -\nabla_{sh,1}(\theta) &> \vec{d}_3 \cdot -\nabla_{sh,1}(\theta),\\
\vec{d}_2 \cdot -\nabla_{sh,2}(\theta) &> \vec{d}_3 \cdot -\nabla_{sh,2}(\theta),\\
\vec{d}_2 \cdot -\nabla_{sh,3}(\theta) &< \vec{d}_3 \cdot -\nabla_{sh,3}(\theta),
\end{aligned}$$

being necessary to find a trade-off between the objectives. These solutions are known as Pareto optimal solutions (dark gray region in Fig. 2).

3.2. Pareto descent directions

.

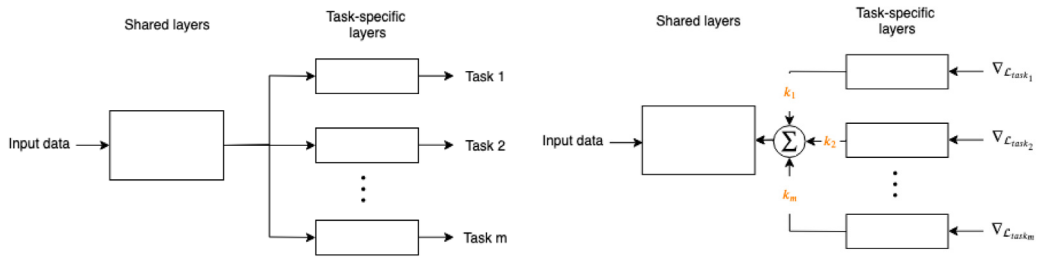
Definition 1. [Pareto-optimality] A decision vector $\mathbf{x}^* \in S$ is Pareto optimal if another decision vector $\mathbf{x} \in S$ does not exist, such that:

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

A Pareto optimal solution can be expressed as a convex combination of the negative gradients ([27,10]). In [30], the notion of Pareto-stationarity was introduced as a necessary condition for Pareto-optimality:

Definition 2 (*Pareto-stationarity*). Given M differentiable objective functions $\mathscr{L}_m(\theta), m = 1,\cdots,M,\ \theta \in \mathrm{IR}^{\mathrm{d}}$, a point $\hat{\theta}$ is considered a Pareto-stationary point, if and only if a convex combination of the gradients that is equal to zero exists, i.e.:

$\sum_{i=1}^{M} k_m \nabla_{sh,m}(\hat{\theta}) = 0$, where $\sum_{i=1}^{M} k_m = 1, k_m \geqslant 0 \quad \forall m$.



(a) Feed-forward (b) Back-propagation

Fig. 1. Multi-task learning architecture. The output from the shared layers is split into a set of m task-specific layers. During back-propagation, 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.

Therefore, if a given point $\hat{\theta}$ is not Pareto-stationary, it will be a descent direction for all objective functions ([31,30]).

Based on Definition 2, a common descent direction can be defined by searching for weighting coefficients that minimize the norm of the convex combination ([2,30]):

$$\text{argmin}_{k_1,\cdots,k_M}\left\{\left|\left|\sum_{\text{m=1}}^{\text{M}} \text{k}_\text{m}\nabla_{sh,\text{m}}\right|\right|_2^2 \quad \sum_{m=1}^{M} k_m = 1,\, k_m \geqslant 0 \quad \forall m\right\}. \tag{2}$$

This solution will result in the minimal-norm element in the convex hull of the negative gradients. However, when we deal with problems of different natures in multi-task learning, we may also have to deal with variations in the order of gradients' magnitude. Thus, when a task is given more importance than another during training, the final model's performance can be negatively impacted. Although the descent direction found by Eq. 2 is based on an optimality condition (Definition 1), this solution may not be representative for all tasks when the tasks' losses are not balanced. In this case, the solution will tend to prefer the direction with the least norm (red vector in Fig. 3).

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. Eq. 2 can be rewritten as:

$$\text{argmin}_{k_1,\cdots,k_M}\left\{\left|\left|\sum_{\text{m=1}}^{\text{M}} \text{k}_\text{m}\bar{\nabla}_{sh,\text{m}}\right|\right|_2^2 \middle| \sum_{m=1}^{M} k_m = 1,\, k_m \geqslant 0 \quad \forall m\right\}. \tag{3}$$

Solving optimization problem (3), the common descent direction can be obtained by a convex combination of each task's gradients 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 Fig. 3), 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. It can be done using a scale factor γ as proposed in [12]. Therefore, the resulting common descent direction, $\vec{v}$, that lies in the central region between tasks can be defined as:

$$\vec{v} = \gamma\sum_{m=1}^{M} k_m\bar{\nabla}_{sh,m}, \tag{4}$$

where $\gamma = \left(\sum_{m=1}^{M}\frac{k_m}{\|\nabla_{sh,m}\|}\right)^{-1}$. 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. This means that, these methods do not consider the history or any indication that the directions defined during the training process were being representative for all tasks. Because of this, the direction found will tend to prefer a specific descent direction (i.e., the direction that results in lesser or greater magnitude, central direction, etc.). 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 T iterations and adopt the idea of tensioners that pull the common descent direction for the task that is diverging.

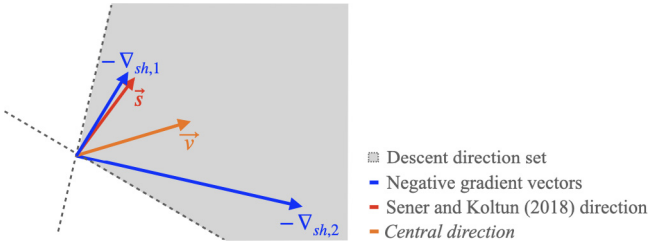


Fig. 3. Common descent directions considering the negative gradient vectors of $\nabla_{sh,1}$ and $\nabla_{sh,2}$. Best visualized with color.

3.3. Task tensioner

Given the central direction, $\vec{v}$, obtained through the Eq. 4, the tensions of each task can be computed according to the following equation:

$$u_m = c_m \cdot \frac{\nabla_{sh,m} - \vec{v}}{\|\nabla_{sh,m} - \vec{v}\|}, \quad \forall m = 1,\ldots,M, \tag{5}$$

where $\|\cdot\|$ is l2-norm, and the c_m factor measure 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 (Fig. 4). Therefore, given the gradient vector of each task, the average of gradients' magnitude accumulated during a period of T iterations is calculated by:

$$\zeta_m(t) = \frac{1}{T}\sum_{i=0}^{T-1}\|\nabla_{sh,m}(t-i)\|. \tag{6}$$

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

$$\delta_m = \frac{\|\zeta_m(t)\|}{\|\zeta_m(t-1)\|} + \log_{10}(\mathscr{L}_m). \tag{7}$$

The term $\log_{10}(\mathscr{L}_m)$ aims to penalize tasks that have a 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.

Given the value of δ_m, we can get the tension factor for each task m:

$$c_m = \frac{\alpha}{1 + e^{(-\delta_m \cdot e + e)}} + 1 - \alpha. \tag{8}$$

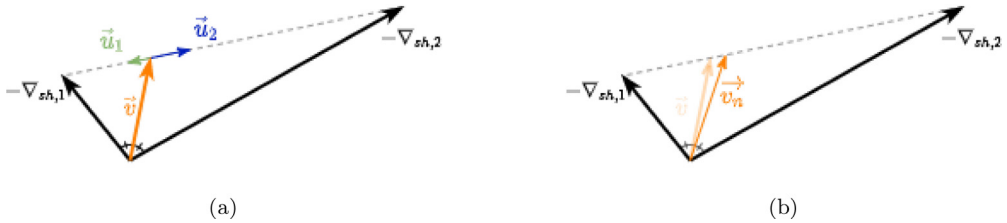
As we want to change the descent direction only if some task is diverging, the c_m values are calculated so that the direction does not change too much from the convergence region of all tasks. Thus, a constant variable, $\alpha \in [0, 1]$, is used to regulate the tension factor's sensitivity. Therefore,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

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 (Eq. 5):

$$\vec{v}_n = \vec{v} + \sum_{m=1}^{M} c_m\left(\frac{\nabla_{sh,m} - \vec{v}}{\|\nabla_{sh,m} - \vec{v}\|}\right), \tag{9}$$

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

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



(a) (b)

Fig. 4. 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.

Algorithm 1 Gradient accumulation

For $i = 0, \ldots, I-1$ **do**
  For $m = 1, \cdots, M$ **do**
    $\mathscr{L}_m \leftarrow \mathscr{L}(f_m(x_i; \theta_{sh}, \theta_m), y_{i,m})$ ▷Compute task-specific loss
    $\nabla_m \leftarrow \frac{\partial \mathscr{L}_m}{\partial \theta_m}$ ▷Gradient descent on task-specific parameters
    $\nabla_{sh,m} \leftarrow \frac{\partial \mathscr{L}_m}{\partial \theta_{sh}}$ ▷Gradient descent on shared parameters
  end for
  Solve (3) to obtain k_m and find common descent direction $\sum k_m \bar{\nabla}_{sh,m}$
  if $i \geqslant T$ **then**
    Compute new common descent direction using Algorithm 2.
  else
    $\vec{v}_n \leftarrow \vec{v}$
  end if
  Update θ_{sh} w.r.t $\vec{v}_n$ with chosen optimizer;
  Update θ_m, $\forall m$ with chosen optimizer.
end for

Algorithm 2 Compute task tension

Require: $i \geqslant T$
for $m = 1, \cdots, M$ **do**
  Compute relative change, δ_m (7)
  Compute tension factor, c_m (8)
  Compute task tension, u_m (5)
end for
$\vec{v}_n \leftarrow \vec{v} + \sum_{m=1}^{M} u_m$ ▷Update common descent direction (9)
return $\vec{v}_n$

4. Experiments

This section presents ablation studies to analyze the behavior of different descent direction methods in problems of multiple logical operators and then evaluates the proposed method in the image domain for multi-task learning. For ablation studies, a simple and narrow Multi-layer Perceptron (MLP) was used to ensure no influence of the network architecture or other variables. Next, for the evaluation experiment, a Convolutional Neural Network was used to classify and reconstruct images from the MNIST dataset adapted to perform multi-task learning.

4.1. Experimental setup and metrics

The experiments were conducted using a machine with Intel Core i7-10750H, 16 GB RAM, and RTX 2060 6 GB.

For quantitative evaluations, we used the mean squared error (MSE) and classification accuracy:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2,$$
$$Accuracy = \frac{TP+TN}{TP+FP+FN+TN},$$

where $\hat{y}$ is the estimated value, y is the ground truth, 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.

4.2. Ablation studies

In these ablation studies, the simultaneous learning of the logical operators XOR and AND was evaluated. We adopted this study because we know exactly how many neurons are needed to generate the hyperplanes that separate the sets of tasks' data in the feature space.

Thus, we eliminate external influences that generate uncertainty if it is not possible to obtain a better result due to the architecture or problem modeling, allowing to evaluate only the descent direction methods. Furthermore, the problems are both complementary and conflicting. This is because, in the 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. Fig. 5a shows a MLP with two shared neurons and two task-specific output neurons. The hyperplanes necessary to separate the data of both tasks are shown in Fig. 5b.

The experiments were performed considering the common descent directions computed according to the methods of uniform weighting, minimum-norm ([2]), central direction ([12]), and gradient modification with tasks' tensors (proposed). To analyze the descent methods considering tasks with different orders of magnitude, the annotations of XOR task were multiplied by a constant $c \geqslant 1$ to change the problem's scale. Each model was trained using the 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 $lr = \{5e-5, 1e-4, 5e-3, 1e-3, 5e-2, 1e-2\}$ of learning rates and the Stochastic Gradient Descent (SGD) optimization algorithm with momentum. Thus, the reported results are the models that best separate the input data.

The Influence of T and α: To verify the behavior of using tensors 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 Eq. 4. 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

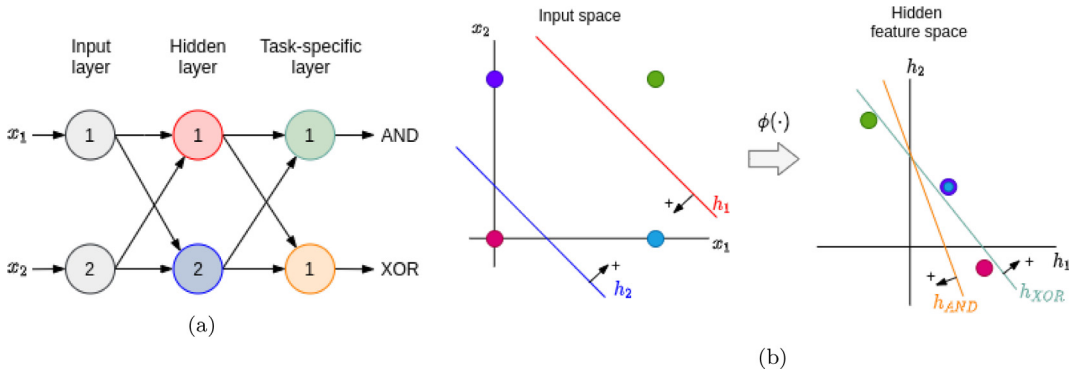


Fig. 5. (a) MLP for AND and XOR tasks. (b) Feature space and hyperplanes necessary to separate the input data. Best visualized with color.

other hand, by adopting a very large value for T, we may be analyzing more information than necessary.

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 direction of descent will be the direction defined by Eq. 4. 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 (Fig. 6). 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 order of magnitudes ($c = 10$). 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 Eq. 7, 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 tensor can be moderated by setting an intermediate value of α. Therefore, considering the analysis presented on this study, we set $T = 10$ (Eq. 6) and $\alpha = 0.3$ (Eq. 8) to perform the following experiments presented in this section.

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 it 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 [2], which has a slightly slower convergence compared to other methods because it prioritizes the task direction with a lower gradient magnitude.

Regarding using the tensors 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 it can be seen, the proposed method resulted in faster convergence. In the case of the central direction, the use of tensors reduced the number of epochs needed from 574.44 to 443.02, resulting in the method that obtained the fastest convergence. And when applied the tensors in the Sener method ([2]), 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 the one 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.

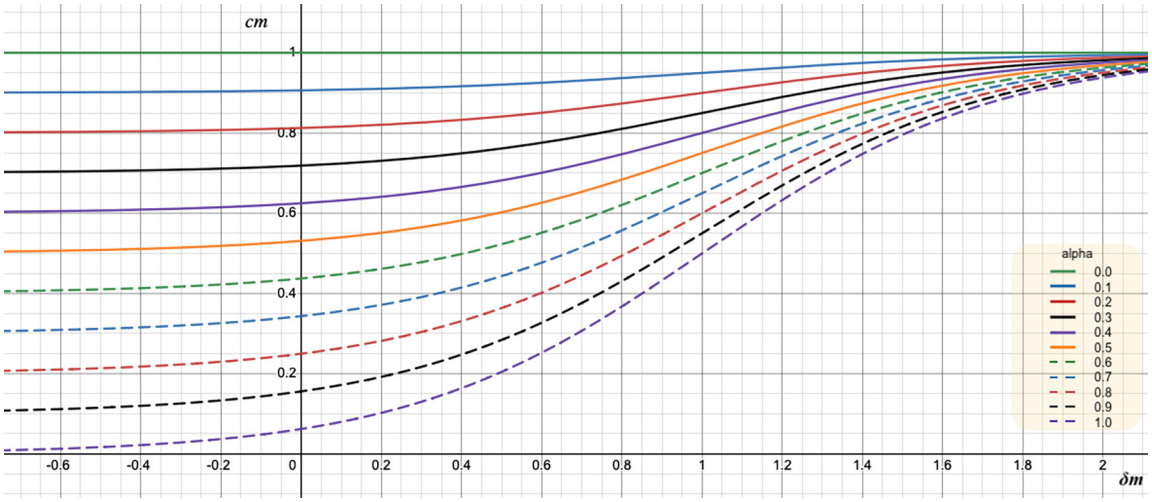


Fig. 6. Behavior of the tensor factor w.r.t. the relative variation of the gradient's norm given the α values (Eq. 8).

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	Convergence rate	Training epoch avg.	Convergence rate	Training epoch avg.	Convergence rate	Training epoch avg.
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	Convergence rate	Training epoch avg.	Convergence rate	Training epoch avg.	Convergence rate	Training epoch avg.
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

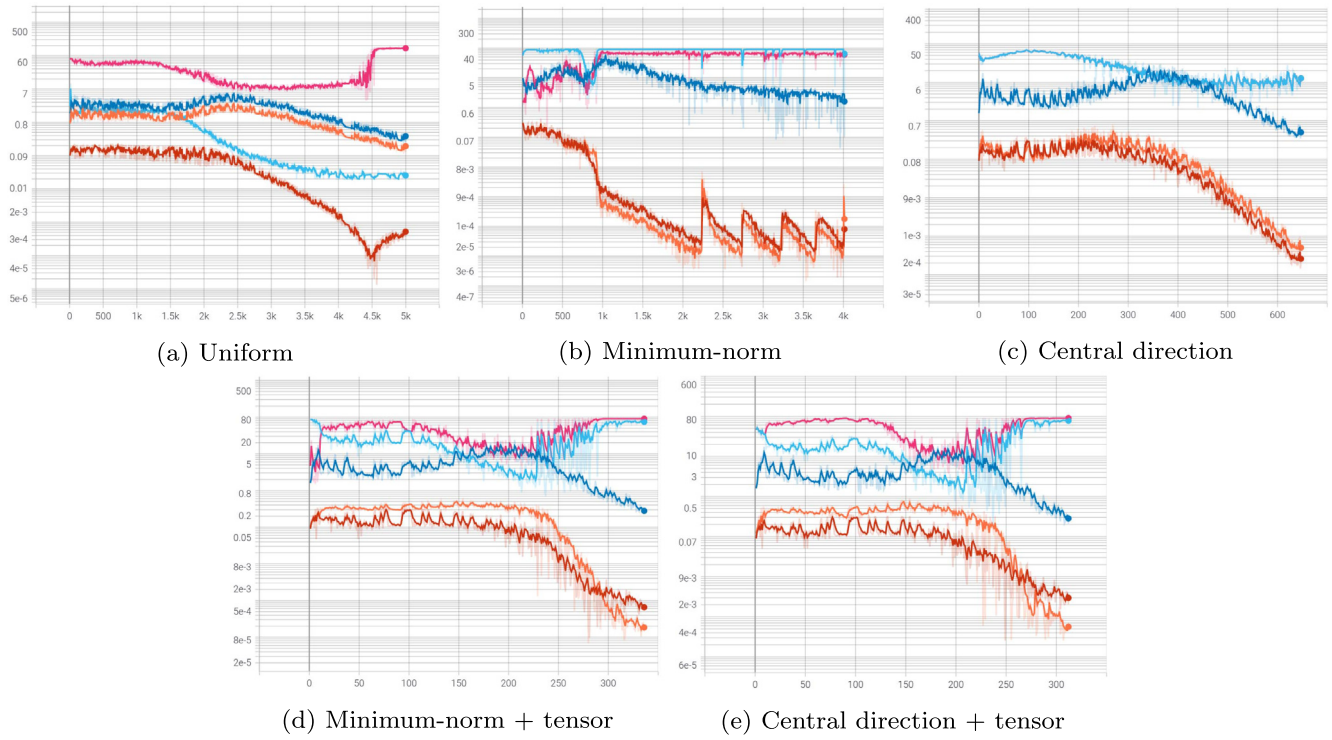
Table 3
The average performance of evaluated methods without changing the problems' scale.

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	$\mathbf{443.02 \pm 21.14}$

The behavior of each descent direction method can be observed in the learning curves of Fig. 7. When uniform weighting is used, we can see that the common 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 the Fig. 7a). This can make the simultaneous learning to be substantially accomplished in relation to a single task, while the common descent direction might not be feasible for the other tasks. As it occurs in the case of the AND task, the angle between its direction and the defined common descent direction is greater than 90 degrees in some iterations (Definition 1). On the other hand, when we have tasks with differences in the gradient scale, the method proposed by [2] 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 (Fig. 7b).

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 Fig. 7c). Therefore, this direction tends to give equal importance to all tasks, regardless of the gradients norm. Figs. 7d and 7e show the minimum-norm and central directions changed with the use of tensors, 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 higher XOR task norm, it increases significantly compared to another task (dark blue curve in the Fig. 7e). From $T = 10$ iterations, the use of tensors starts to dynamically change the common descent direction in relation to the diverging task. 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 Eq. 7. The same behavior can be observed when we apply the use of tensors 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 are shown in Table 4. Since only the scale of the XOR problem was changed, the performance of the model trained separately for 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 performing the simultaneous learning of both tasks, the uniform weighting resulted in the worst performance, being able to separate the feature space in only 46% of the 50 training executions, and, in the best case, needing more than 1,000 epochs for convergence. Regarding the methods of central direction and minimum-norm ([2,12]), both converged in all training executions. However, as it can be seen in the fourth row of Table 4, the method proposed by [2] needs more iterations. Finally, when we use the tensors to change the central and the minimum-norm directions, we again improve the convergence of the models, sig-



(a) Uniform (b) Minimum-norm (c) Central direction

(d) Minimum-norm + tensor (e) Central direction + tensor

Fig. 7. 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. Best visualized with color.

Table 4
The average performance of evaluated methods considering $c = 10$ for the XOR task.

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	$\mathbf{188.20 \pm 24.29}$

nificantly reducing the number of epochs compared to the minimum norm method and obtaining the best performance among the evaluated methods.

4.3. 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 ([32]). To turn it into a multi-task learning problem, we use the idea of adapting the MNIST dataset ([33,2]), in which each image is composed of two overlapping digit images (Fig. 8). 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 was classifying 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. Therefore, to change the scale and nature of the problem, we defined the first task as 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 with complementary information, 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 Fig. 9 ([32,2]). The models were trained considering the Stochastic Gradient Descent (SGD) optimization algorithm with momentum and a set $lr = \{1e-3, 5e-3, 1e-2, 5e-2, 1e-1\}$ 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 with a different seed value ten times to get deterministic random data. Furthermore, the training process was interrupted only when there was no improvement in validation metrics in 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 ablation studies, 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 weighting was the method that resulted in the best accuracy for the classification problem (second row). However, in addition to extracting relevant features for classification, the shared encoder must be able to compress the input data (IR^{784}) into a latent space representation (IR^{50}), so that the task-specific decoder can then reconstruct this representation. In this case, since the shared encoder was mainly optimized in relation

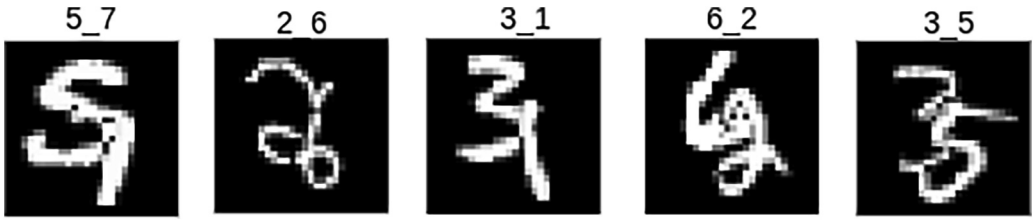


Fig. 8. Images examples from the MNIST dataset modified for multi-task learning. Images have a size of 28 × 28 pixels.

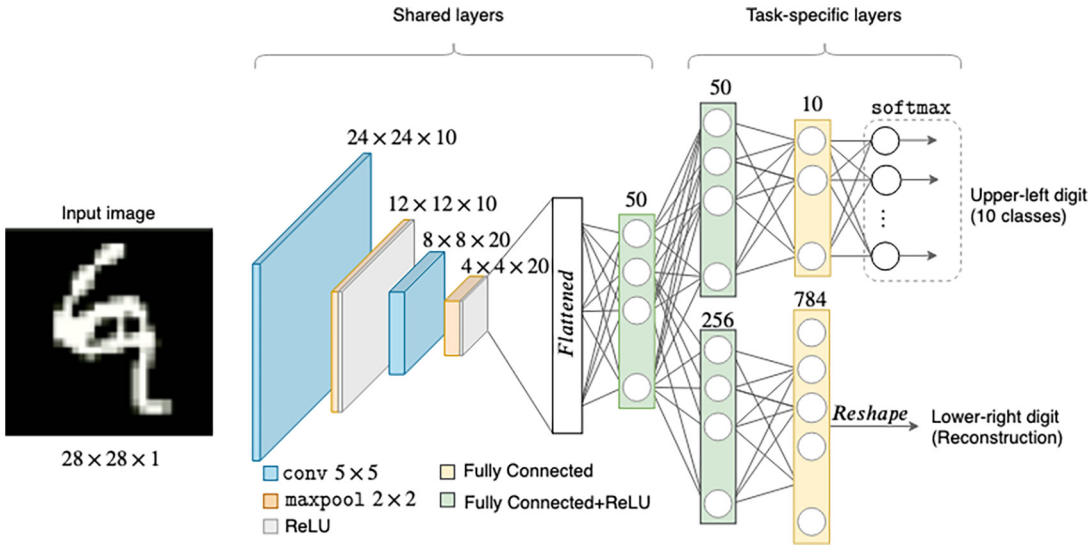


Fig. 9. Multi-task learning architecture for upper-left digit classification and lower-right digit reconstruction. Best visualized with color.

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

to the classification problem, the uniform weighting reconstruction error was greater than the individual task.

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

Finally, the last two rows present the use of tensors 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 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. This can be better observed in Fig. 10, which shows the performance of the ten trained models of each method. To simplify the visualization, we inverted the *y*-axis of the plot. Thus, the methods with the best results are those closest to the upper-right corner. In addition to the achieved performance, the proposed method presents the advantage of reducing the use of computational resources and inference latency. This is because the memory usage and inference time positively correlate to the number of trainable parameters in the network architecture [34]. Therefore, since most of the model parameters are shared between tasks, it becomes computationally more efficient than single-task architectures.

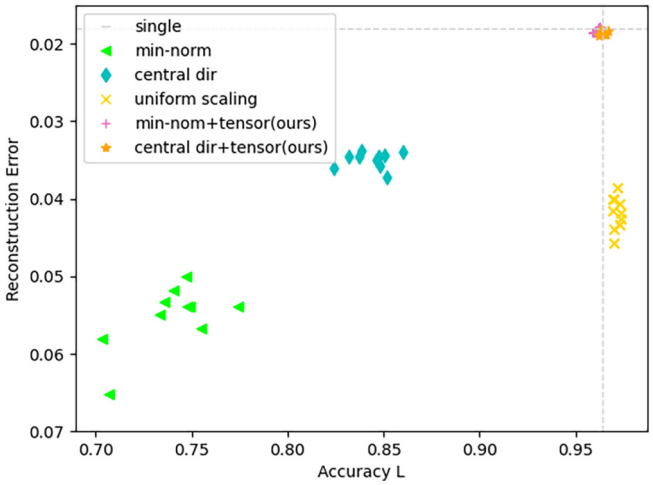


Fig. 10. Plot of all trained models with the average performance of single tasks as baseline. The proposed method resulted in a good balance between tasks, resulting in a performance similar to or even better than single-task models. Upper-right is better. Best visualized with color.

5. Conclusion

In this work, we address the simultaneous learning of conflicting tasks that have differences in the magnitude scale. Our method is motivated by the difficulty in performing simultaneous learning of multiple tasks in a hard parameter sharing approach without harming the performance of tasks. To this end, we propose to dynamically change the common descent direction between multiple tasks considering their convergence behavior, ensuring that the new direction will be feasible for all of them.

Experimental results showed that the proposed method outperforms the state-of-the-art methods in simultaneous learning of conflicting tasks. Thanks to the use of temporal information to properly adjust the importance of each task, it was possible to maximize performance without harming any task. Furthermore, the proposed method provided consistent performance with low variance in the tested problems.

Despite the fact that our 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 average of gradients over the last T iterations 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.

And 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. In addition to multi-task learning, we believe that it is also possible to use the proposed method to explore other areas where it is necessary to find a trade-off between learning multiple tasks, such as to deal 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.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgement

The authors would like to express their gratitude to Luiz Ricardo Takeshi Horita for insightful discussions and feedback. **Funding:** This work was supported by the São Paulo Research Foundation – FAPESP [Grant No. 2019/03366-2] and the Coordination of Improvement of Higher Education Personnel – Brazil – CAPES [Finance Code 001]. This work was also supported in part by the Brazilian National Council for Scientific and Technological Development – CNPq [Grant No. 465755/2014-3], by CAPES [Grant No. 88887.136349/2017–00], and the FAPESP [Grant No. 2014/50851–0].

References

[1] R. Caruana, Multitask Learning (Ph.D. thesis), Carnegie Mellon University, 1997.
[2] O. Sener, V. Koltun, Multi-task learning as multi-objective optimization, in: Advances in Neural Information Processing Systems, Vol. 2018-Decem, 2018, pp. 527–538.
[3] R. Cipolla, Y. Gal, A. Kendall, Multi-task Learning Using Uncertainty to Weigh Losses for Scene Geometry and Semantics, in: 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, Vol. abs/1705.0, IEEE, 2018, pp. 7482–7491.
[4] Z. Chen, V. Badrinarayanan, C.Y. Lee, A. Rabinovich, GradNorm: Gradient normalization for adaptive loss balancing in deep multitask networks, in: 35th International Conference on Machine Learning ICML, 2018, 2018,, pp. 1240–1251.
[5] A.T.M. Nakamura, V. Grassi, D.F. Wolf, An effective combination of loss gradients for multi-task learning applied on instance segmentation and depth estimation, Eng. Appl. Artif. Intell. 100 (2021), https://doi.org/10.1016/j.engappai.2021.104205 104205.
[6] Y. Mao, S. Yun, W. Liu, B. Du, Tchebycheff Procedure for Multi-task Text Classification, in: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, Association for Computational Linguistics, Stroudsburg, PA, USA, 2020, pp. 4217–4226, https://doi.org/10.18653/v1/2020.acl-main.388.
[7] P. Liu, X. Qiu, H. Xuanjing, Recurrent neural network for text classification with multi-task learning, in: IJCAI International Joint Conference on Artificial Intelligence, Vol. 2016-Janua, 2016, pp. 2873–2879. arXiv:1605.05101.
[8] L. Xiao, H. Zhang, W. Chen, 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, Vol. 2, 2018, pp. 726–731. doi:10.18653/v1/n18-2114.
[9] J. Uhrig, M. Cordts, U. Franke, T. Brox, Pixel-level encoding and depth layering for instance-level semantic labeling, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) (2016) 14–25.
[10] K. Miettinen, Nonlinear Multiobjective Optimization, Springer, US, 1998.
[11] M. Ehrgott, Multicriteria optimization, 2nd Edition, Springer-Verlag, Berlin/Heidelberg, 2005. doi:10.1007/3-540-27659-9.
[12] A. Katrutsa, D. Merkulov, N. Tursynbek, I. Oseledets, Follow the bisector: a simple method for multi-objective optimization (2020). arXiv:2007.06937.
[13] R.A. Caruana, Multitask Learning: A Knowledge-Based Source of Inductive Bias, Mach. Learn. Proc. 1993 (1993) 41–48.
[14] Y. LeCun, B. Boser, J.S. Denker, D. Henderson, R.E. Howard, W. Hubbard, L.D. Jackel, Backpropagation Applied to Handwritten Zip Code Recognition, Neural Comput. 1 (4) (1989) 541–551, https://doi.org/10.1162/neco.1989.1.4.541.
[15] Y. Zhang, Q. Yang, A Survey on Multi-Task Learning, IEEE Trans. Knowl. Data Eng. (2021) 1–1 arXiv:1707.08114, doi:10.1109/TKDE.2021.3070203.
[16] S. Vandenhende, S. Georgoulis, B. De Brabandere, L. Van Gool, Branched Multi-Task Networks: Deciding What Layers To Share (apr 2019). arXiv:1904.02920.
[17] I. Misra, A. Shrivastava, A. Gupta, M. Hebert, Cross-Stitch Networks for Multi-task Learning, in: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Vol. 2016-Decem, 2016, pp. 3994–4003.
[18] S. Ruder, J. Bingel, I. Augenstein, A. Søgaard, Latent Multi-Task Architecture Learning, in: Proceedings of the AAAI Conference on Artificial Intelligence 33, 2019, pp. 4822–4829, https://doi.org/10.1609/aaai.v33i01.33014822.
[19] P. Liu, X. Qiu, X. Huang, 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), Vol. 1, Association for Computational Linguistics, 2017, pp. 1–10.
[20] S. Chennupati, G. Sistu, S. Yogamani, S. Rawashdeh, 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, Vol. 5, 2019, pp. 645–652.
[21] M. Teichmann, M. Weber, M. Zöllner, R. Cipolla, R. Urtasun, MultiNet: Real-time Joint Semantic Reasoning for Autonomous Driving, IEEE Intelligent Vehicles Symposium, Proceedings (2018) 1013–1020.
[22] D. Sanchez, M. Oliu, M. Madadi, X. Baro, S. Escalera, 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), IEEE, 2019, pp. 1–8.
[23] S. Liu, E. Johns, A.J. Davison, End-To-End Multi-Task Learning With Attention, in: 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2019, pp. 1871–1880.
[24] C. Li, J. Yan, F. Wei, W. Dong, Q. Liu, H. Zha, Self-Paced Multi-Task Learning, 31st AAAI Conference on Artificial Intelligence, AAAI 2017 (2016) 2175–2181arXiv:1604.01474.
[25] M. Guo, A. Haque, D.A. Huang, S. Yeung, L. Fei-Fei, Dynamic Task Prioritization for Multitask Learning, in, in: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2018, pp. 282–299.
[26] X. Lin, H.-L. Zhen, Z. Li, Q.-F. Zhang, S. Kwong, Pareto multi-task learning, in: H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, R. Garnett (Eds.), Advances in Neural Information Processing Systems, Vol. 32, Curran Associates Inc, 2019.
[27] K. Harada, J. Sakuma, S. Kobayashi, Local search for multiobjective function optimization: Pareto descent method, in: GECCO 2006 – Genetic and Evolutionary Computation Conference, vol. 1, 2006, pp. 659–666.
[28] D. Mahapatra, V. Rajan, Multi-Task learning with user preferences: Gradient descent with controlled ascent in pareto optimization, in: 37th International Conference on Machine Learning, ICML 2020, Vol. Part F16814, 2020, pp. 6553–6563.
[29] J. Fliege, B.F. Svaiter, Steepest descent methods for multicriteria optimization, Math. Methods Oper. Res. 51 (3) (2000) 479–494, https://doi.org/10.1007/s001860000043.

[30] J.A. Désidéri, Multiple-gradient descent algorithm (MGDA) for multiobjective optimization, C.R. Math. 350 (5–6) (2012) 313–318, https://doi.org/10.1016/j.crma.2012.03.014.
[31] S. Schäffler, R. Schultz, K. Weinzierl, Stochastic method for the solution of unconstrained vector optimization problems, J. Optim. Theory Appl. 114 (1) (2002) 209–222, https://doi.org/10.1023/A:1015472306888.
[32] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proc. IEEE 86 (11) (1998) 2278–2323, https://doi.org/10.1109/5.726791.
[33] S. Sabour, N. Frosst, G.E. Hinton, Dynamic routing between capsules, in: Advances in Neural Information Processing Systems, Vol. 2017-Decem, 2017, pp. 3857–3867. arXiv:1710.09829.
[34] S. Bianco, R. Cadene, L. Celona, P. Napoletano, Benchmark analysis of representative deep neural network architectures, IEEE Access 6 (2018) 64270–64277, https://doi.org/10.1109/ACCESS.2018.2877890.

Angelica Tiemi Mizuno Nakamura is a PhD student in the Institute of Mathematics and Computer Science at the University of São Paulo (ICMC/USP). She obtained her MS degree in Electrical Engineering at the University of São Paulo in 2017. Her current research interests are Machine Learning, Computer vision and Autonomous Vehicles.

Valdir Grassi Jr. received a Ph.D degree in Mechanical Engineering from the University of São Paulo, São Paulo, Brazil, in 2006. He is currently an Associate Professor at the São Carlos School of Engineering of the University of São Paulo (EESC/USP), Brazil. His main research interests include autonomous vehicle navigation, motion planning and control, computer vision applied to robotics, and machine learning.

Denis Fernando Wolf received a PhD degree in Computer Science at the University of Southern California – USC in 2006. He is currently an Associate Professor in the Department of Computer Systems at the University of São Paulo (ICMC-USP). His current research interests are Mobile Robotics, Machine Learning, Computer Vision and Embedded Systems.