Report on Student Research Project

Second Year of French Grande École

ENSTA Paris - Institut Polytechnique de Paris

Lipschitz Regularization for Neural Networks in High Dimension

Authored by :

Mr. Raphaël BERNAS - Student at ENSTA Paris in Applied Mathematics

Supervised by :

Dr. Pr. Martin BURGER and Dr. Tim Roith

From May 13th, to August 17th, 2024

Dr. Pr. Éric Lunéville	ENSTA Paris	- Internship Tutor and Examiner
Dr. Pr. Laurent BOURGEOIS	ENSTA Paris	- Examiner

Internship at Deutsches Elektronen-Synchrotron (DESY)

Notkestrasse 85, 22607 Hamburg, Germany

Class of : 2022/2025

Defended on August 28th, 2024.

Non - Confidential

Confidentiality note

This report is non-confidential, thus it can be shared online and kept by ENSTA Paris.

Contents

Acknowledgments

All the works presented here are the results of a simultaneous research lead by Dr. Tim Roith, Dr. Pr. Martin Burger, Dr. Pr. Leon Bungert, Dr. Kehan Shi and Student Raphaël Bernas.
These works mostly rely on [Bun22].

Personal Thanks

I would like to express my deepest gratitude to the Computational Imaging Group and Helmholtz Imaging at DESY for their warm welcome.

I'm specially thankful to Pr. Burger for giving me this opportunity. To Tim for all the things he taught me during this journey, they are invaluable advices. To Leon with whom researching was very gratifying. To Kehan whom took time to explain hyper-graphs theory to me. To Danielle, Jin, Lorenz, Lukas, Samira, Yi and all members of the imaging group that welcomed me warmly.

I would like to express my deepest gratitude to my family and my school, ENSTA Paris (with the Institut Polytechnique de Paris), for their support.

Finally, I would like to once again express my gratitude to DESY and specially to Ieva Gilyte-Robertson, without whom this internship would not have been possible.

Information

The work presented in this report may lead to the publication of a paper, which would be co-authored by Leon Bungert, Martin Burger, Kehan Shi, Tim Roith, and myself.

If you want a section that explain the content of this report with a non-mathematics approache, please refer to Section 7.5.

It is highly recommended to refer to Section 7.1 **which detail notation and** Section 7.2 **which detail terms employed.**
See those sections when there is a word or a notation you do not understand.

The code used for all experiments conducted in the main part of this report can be found on :

https://github.com/TimRoith/CLIP.

The GitHub is currently divided into three branches :

- *main* : This branch contains the code used in [Bun22] which introduces the CLIP algorithm.
- *optwrite* : This branch contains the code used for our experiments on CLIP (1D experiments and optimizers).
- *newstruct* : This is the most important branch for this report, it contains the FLIP package and our experiments.

Résumé

La robustesse des réseaux de neurones est un enjeu crucial dans le domaine de l'apprentissage automatique, impactant diverses applications. De petites perturbations dans les données d'entrée peuvent affecter de manière significative la qualité des sorties, ce qui entraîne des problèmes en classification d'images, dans les véhicules autonomes, et en détection d'objets. Alors que l'apprentissage profond continue de croître en importance, ces problèmes pourraient représenter des risques de sécurité pour les systèmes reposant sur des réseaux neuronaux profonds. Pour remédier à cela, des méthodes de régularisation ont été développées afin d'assurer la robustesse des réseaux neuronaux profonds (DNN). L'une de ces méthodes consiste à pénaliser le modèle en fonction de sa constante de Lipschitz. Cependant, le calcul de cette constante est computationnellement infaisable, en particulier pour des dimensions élevées. Une méthode proposée par *Bungert et al* - dans CLIP : Cheap Lipschitz Training of Neural Networks - suggère de calculer la constante de Lipschitz sur un ensemble restreint déterminé par des méthodes adversariales. Ce travail vise à étendre les résultats trouvés par *Bungert et al* et à relever les défis rencontrés dans des dimensions plus élevées.

Mots-clés - Constante de Lipschitz, Entrainement adversariale, Réseaux neuronaux, Robustesse, Sur-apprentissage, Haute dimension.

Abstract

Neural network robustness is a critical issue in the field of machine learning, affecting various applications. Small perturbations in the input can significantly impact the output quality, leading to problems in image classification, autonomous vehicles, and object detection. As deep learning continues to grow in importance, these issues could lead to security risks for systems relying on deep neural networks. To address this, regularization methods have been developed to ensure Deep Neural Network (DNN) robustness. One such method involves penalizing the model based on its Lipschitz constant. However, calculating this constant is computationally unfeasible, especially in higher dimensions. A method proposed by *Bungert et al* - in CLIP : Cheap Lipschitz Training of Neural Networks - suggests computing the Lipschitz constant on a restricted set determined by adversarial methods. This work aim to extend the results found by *Bungert et al* and address the challenges encountered in higher dimensions.

Keywords - Lipschitz Constant, Adversarial Training, Neural Network, Robustness, Over-fitting, High Dimension.

1 Introduction

To be trained, Deep Neural Networks (DNN) models[1] rely on a substantial amount of data. This dependence on training data presents several drawbacks, one of which is over-fitting. Over-fitting occurs when the model prioritizes fitting the training data perfectly over producing generalizable results, thereby reducing its robustness against adversarial[2] attacks [Goo15] which is a slight perturbation on the input that leads the model to makes errors. To address this, various methods such as adversarial training [Mad18] have been developed. The primary idea behind these methods is to train the model with adversarial perturbed data to enhance its robustness against such attacks. While adversarial training significantly improves model robustness, regularization can also be employed to increase DNN robustness against adversarial attacks.

Let us define our model as the function f_θ, where $\theta \in \Theta$ represents the parameters of the DNN[3], and the loss function[4] such that for a set of data ($\Gamma \subset Z = \mathcal{X} \times \mathcal{Y}$), where $\mathcal{X}$ and $\mathcal{Y}$ are the sets of all the values our input $x \in \mathcal{X}$ and our output $y \in \mathcal{Y}$ can take, we aim to solve the following problem:

$$\min_\theta \sum_{(x,y)\in\Gamma} \text{loss}(f_\theta(x), y) \tag{1}$$

To enhance the robustness of the solution, we introduce a regularization term *reg*, leading to the following problem:

$$\text{Let } \lambda \in \mathbb{R}_+,$$

$$\min_\theta \left(\sum_{(x,y)\in\Gamma} \text{loss}(f_\theta(x), y) + \lambda\, \text{reg}(f_\theta) \right) \tag{2}$$

It is important to understand that *reg* is a function which depends on a given f_θ. Thus, for our current problem it only depends on θ. The weight λ helps to calibrate between the standard training in Problem (1) and the added regularization

[1]For more details on DNN, robustness and regularization please check [Bos20]

[2]In Section 7.3, find a detailed explanation on adversarial attacks and training.

[3]For example a fully connected model takes the following form : given an activation function σ for instance $\sigma(x) = \frac{1}{1+e^{-x}}$ the sigmoid and a list of matrices of different sizes $(W_i)_{i\in[|1,N|]}$, we have $f_\theta(x) = \sigma(W_N\sigma(...\sigma(W_1x)))$ such that here $\theta = (W_i)_{i\in[|1,N|]}$.

[4]A *loss* function is used to compare a model output with data values. For example a famous loss function is the Mean Squared Error (MSE) that is computed as follows : $MSE(f_\theta, \Gamma) = \frac{1}{|\Gamma|}\sum_{(x,y)\in\Gamma}\|f_\theta(x) - y\|^2$ (Average distance between model prediction and true data value).

Algorithm 1: CLIP (Cheap Lipschitz Training)

```
input : $\mathbf{E}$, number of epochs. $\mathbf{\Gamma}$, data set. $\lambda$, initial value of the regularizer weight.
for epoch $e = 1$ to $E$ do
    for minibatch $B \subset \Gamma$ do
        $\mathcal{X}_{lip} \leftarrow \text{SGDM}_\tau(\text{Lip}(f_\theta, \cdot), \mathcal{X}_{lip})$
        $\theta \leftarrow \text{SGDM}_{\eta,\gamma}\left(\frac{1}{|B|}\sum_{(x,y)\in B} loss(f_\theta(x), y) + \lambda\ Lip(f_\theta, \mathcal{X}_{lip})\right)$
        $accuracy \leftarrow accuracy + \frac{1}{|B|}\sum_{(x,y)\in B} \mathbf{1}(x,y)_{\{f_\theta(x)=y\}}$
        if $accuracy > \alpha$ then
            $\lambda \leftarrow \lambda + d\lambda$
        else
            $\lambda \leftarrow \lambda - d\lambda$
```

term.

Regularization is a method that permits to enforce certain properties in a model. One type of regularization that can be used is based on the Lipschitz constant. The Lipschitz constant is defined as follows, for a given function f defined on a set $\mathcal{X}$:

$$Lip(f, \mathcal{X}) = inf\{L \in \mathbb{R}_+,\ \|f(x) - f(x')\| \leq L\|x - x'\|,\ \forall x, x' \in \mathcal{X}\} \in \overline{\mathbb{R}}_+ = \mathbb{R}_+ \cup \{+\infty\}$$

The Lipschitz constant provides the maximum bound of a double cone within which the function's curve remains confined for any given point. Thus, it only depends on the function for which it is computed and the set where this function is defined. Regularizing with this constant reduces the overall variation of our model, and we expect it to reduces over-fitting.

There are various ways to compute this constant, but as explained in [Sca18], the basic computation is NP-hard and therefore not feasible. The regularization used in [Bun22] is:

$$\text{Lip}(f_\theta, \mathcal{X}_{\text{lip}}) = \sup_{x,x' \in \mathcal{X}_{\text{lip}}} \frac{||f_\theta(x') - f_\theta(x)||}{||x' - x||} \tag{3}$$

where $\mathcal{X}_{\text{lip}} \subset \mathcal{X}^2$ is defined as a set that is updated at every iteration following a gradient ascent process that increases the Lipschitz constant computed on the subset.

This special definition of Lipschitz constant estimation distinguishes the proposition in [Bun22] from the approaches commonly used in other papers such as [Lat20], where the computation is an approximation designed to upper bound the Lipschitz constant value.

The training process in [Bun22] follows Algorithm 1 in which *SGDM* is a gradient descent method that finds a local minimum for a given formula. The idea is to compute the subset $\mathcal{X}_{\text{lip}}$ for which the Lipschitz constant value is locally the highest. Then, a gradient descent step is performed on the parameters to minimize the total loss and the Lipschitz regularization.

Main contribution: The main contributions of our work are the improvements of the *CLIP*-algorithm computation time and method and the proposal of a new Lipschitz regularization based algorithm which we study here.

This report unfolds as follows: Firstly, we test the CLIP algorithm on a 1D problem to verify its efficiency and ensure there are no potential problems we might overlook. Next, we improve CLIP and conduct tests in higher dimensions. In the process, we discovered and tested a new regularizer, which is part of the new FLIP regularization family that we define in this work. This ultimately leads us to developing an equivalent of total variation regularization for neural networks. Finally, we will theoretically study this new regularizer.

2 1D Experimentation

The results obtained in [Bun22] for the $CLIP$-Algorithm 1 have shown a significant increase in model accuracy when confronted with adversarial attacks. These results were achieved using the well-known datasets MNIST [LeC98] and Fashion-MNIST [Xia17], which are high-dimensional data. The objective of this section is to assess the quality of those results through a **visual inspection** in 1D.

2.1 Polynomial Approximation

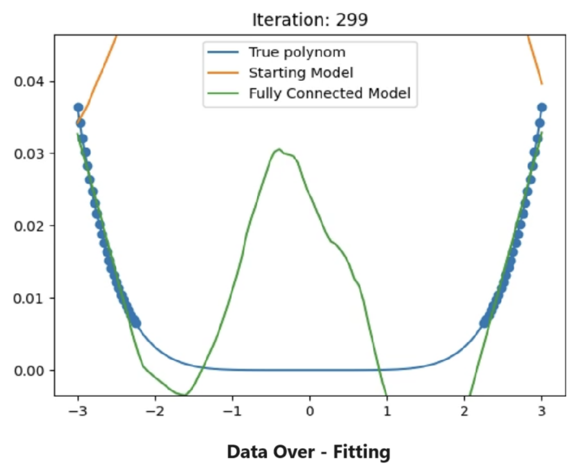
It is important to note that neural networks are not typically designed to approximate polynomial functions and are rarely the optimal method for approximating a curve in a 2D plane. However, we will perform such approximations because they facilitate interpretation.

We proceed to obtain a set of data based on a polynomial function $P = \epsilon(0.5 + 0.01X + X^6)$ where ϵ is a scaling value. We compute $CLIP$ for a fully connected model. For a correctly made dataset with no missing data, we obtain similar results with or without the regularization. While the Lipschitz constant is indeed reduced in the penalized model, indicating increased in local robustness, this improvement is not visually apparent in this scenario.

Next, we assume that a portion of the data is missing. In this case, computing the model with and without Lipschitz regularization does have a significant visual impact. Specifically, the over-fitting of the non-penalized model leads to an incorrect approximation, whereas the penalized model appears closer to the original polynomial function. These results are illustrated in Figure 1. In both images, the blue line refers to the polynomial function we want to approximate. The blue dots are our dataset Γ. The green line refers to the neural network's approximated polynomial function. We clearly observe that with Lipschitz regularization, we are more accurate in the regions where there is less data. Indeed, the steep part for the standard trained model is what we call over-fitting—the model tries to fit the data perfectly without regard for the regions where there is no data.

What has been done in Figure 1 is something we cannot expect to be capable of computing in higher dimensions. Indeed, in Figure 1, the Lipschitz regularized model was calculated with a significant number of iterations for the $\mathcal{X}_{\text{lip}}$ update, ensuring a robust Lipschitz constant. However, since these iterations are necessary for every batch, it quickly becomes unfeasible in higher dimensions. When approaching a more realistic process, the solution appears as shown in Figure 2.

We observe from Figure 2 that the regularization is dependent on the data distribution. Specifically, in regions where the data are scattered, the Lipschitz constant approaches zero, while in other regions, the model does not significantly account for it. This phenomenon can be explained as follows: $\mathcal{X}_{\text{lip}}^{\text{initialization}}$ is a subset of $\Gamma_{\mathcal{X}}$, where $\Gamma_{\mathcal{X}}$ is the set of values that the data assumes in $\mathcal{X}$. If a local maximum exists in $\Gamma_{\mathcal{X}}$ for the Lipschitz constant, or if there is a bias caused by missing data, the estimation of the Lipschitz constant becomes significantly lower. Indeed, the gradient ascent is then



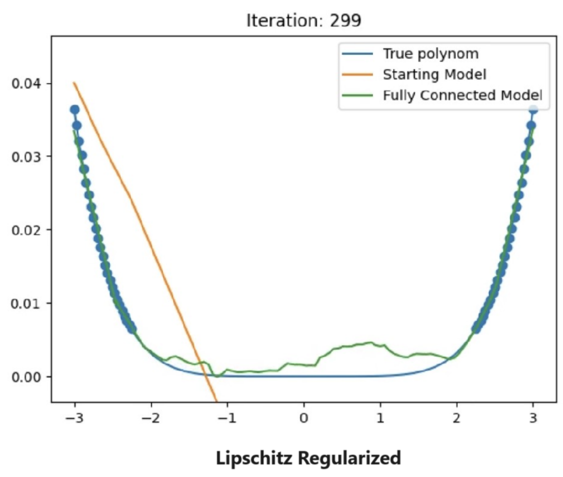


Figure 1: DNN fitting polynomial function - Over-fitting VS Lipschitz regularization (Training video no space: without Lipschitz and with Lipschitz)

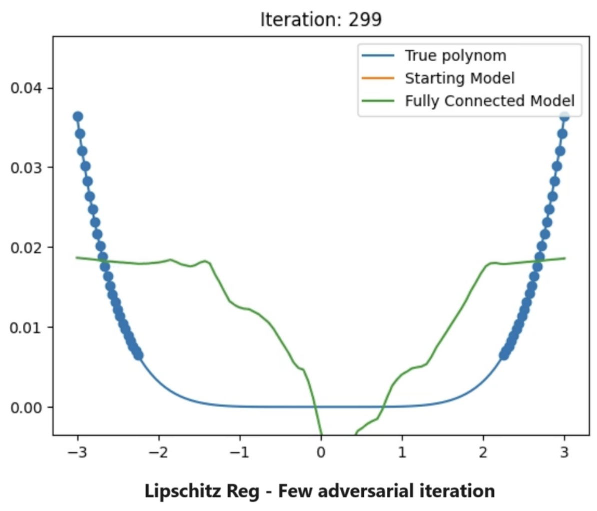


Figure 2: DNN fitting polynomial function - Lipschitz regularization with few X_{lip} update (Training video : here)

stuck. This elucidates why the model is nearly constant at the boundary of $\mathcal{X} = (-3, 3)$, and why it appears that the Lipschitz constant is significantly higher in the remaining regions. But thanks to this observation in 1D we discovered an issue that would have been difficult to observe in higher dimension. Note that processing MNIST data is a more suitable application for a DNN and that these data do not suffer significantly from missing parts.

2.2 Optimizers comparison

In [Bun22], the authors used a gradient descent/ascent algorithm, which yield good results. However, it could be interesting to test other gradient-based optimization algorithms, such as the Nesterov Accelerated Gradient[5] (Algorithm 2) or the Adam Optimizer [Kin15]. We would then change the way we compute the $\mathcal{X}_{lip}$ set as follow for NAG and Adam :

$\mathcal{X}_{lip} \leftarrow \text{NAG}_{\eta}(Lip(f_\theta, \cdot), \mathcal{X}_{lip})$

$\mathcal{X}_{lip} \leftarrow \text{Adam}_{\eta,\beta_1,\beta_2}(Lip(f_\theta, \cdot), \mathcal{X}_{lip})$

[5]There exist multiple similar definitions for Nesterov Accelerated Gradient algorithm. The one given here is defined in an ENSTA Paris course named OPT202 presented by Professor Simonetto.

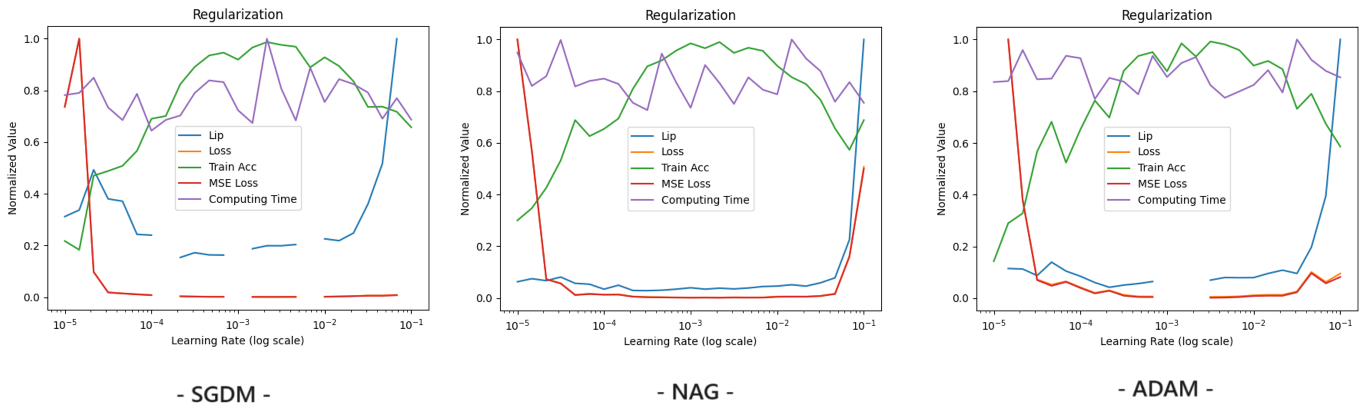


Figure 3: Comparison of SGDM, NAG, ADAM results for log scaled learning rate - SGDM $time^{computation}_{maximal} = 0.0185\,s$ - NAG $time^{computation}_{maximal} = 0.0173\,s$ - ADAM $time^{computation}_{maximal} = 0.0176\,s$ -

Algorithm 2: Nesterov Accelerated Gradient (NAG)

```
Input  : Initial parameters $\theta_0$, learning rate $\eta$, number of iterations $N$
Output: Optimized parameters $\theta$
Initialize $\nu_0 \leftarrow \theta_0$
Initialize $\lambda_0 \leftarrow 0$
for $k = 0$ to $N - 1$ do
    $\theta_{k+1} \leftarrow \nu_k - \eta\nabla_\nu Loss(f_{\nu_k}(\cdot), \cdot)$
    $\nu_{k+1} \leftarrow \gamma_k\theta_k + (1 - \gamma_k)\theta_{k+1}$
    $\lambda_{k+1} \leftarrow \frac{1+\sqrt{1+4\lambda_k^2}}{2}$
    $\gamma_{k+1} \leftarrow \frac{1-\lambda_k}{\lambda_{k+1}}$
```

Subsequently we proceed to test the optimizers with different learning rate values. For each learning rate η and for each optimizer, we train 10 fully connected models and then take the mean values of their outputs (GPU: Nvidia GeForce RTX 3050 Laptop). This gives us Figure 3, in which we can check the accuracy, shown by the green line, and the normalized computing time, shown in purple. The missing parts in the loss values are due to some models which for some learning rates returned values that are either too small or too large to be processed by PyTorch, resulting in "NaN" values. What is interesting here is the evolution of accuracy and computing time. From the results we obtained, we can see that while Nesterov is the quickest because it takes the lowest computation time, it is often less accurate than the others. SGDM, on the other hand, shows great accuracy over a larger range and is more stable, but slower. ADAM represents a balanced solution between these two. We can still conclude that these results are close to each other, allowing us the freedom to choose the optimizer.

2.3 Lambda evolution

It was remarked in [Bun22] that for $\lambda \longrightarrow +\infty$ and with an Euclidean loss (MSE), our model $f_\theta \longrightarrow \frac{1}{|\Gamma|}\sum_{(x,y)\in\Gamma} y$. This means the model converges to the mean of the data values. Using our 1D representation, we can confirm the proven result. Indeed, Figure 4 shows that the model tends to the mean of the data values for increasing λ.

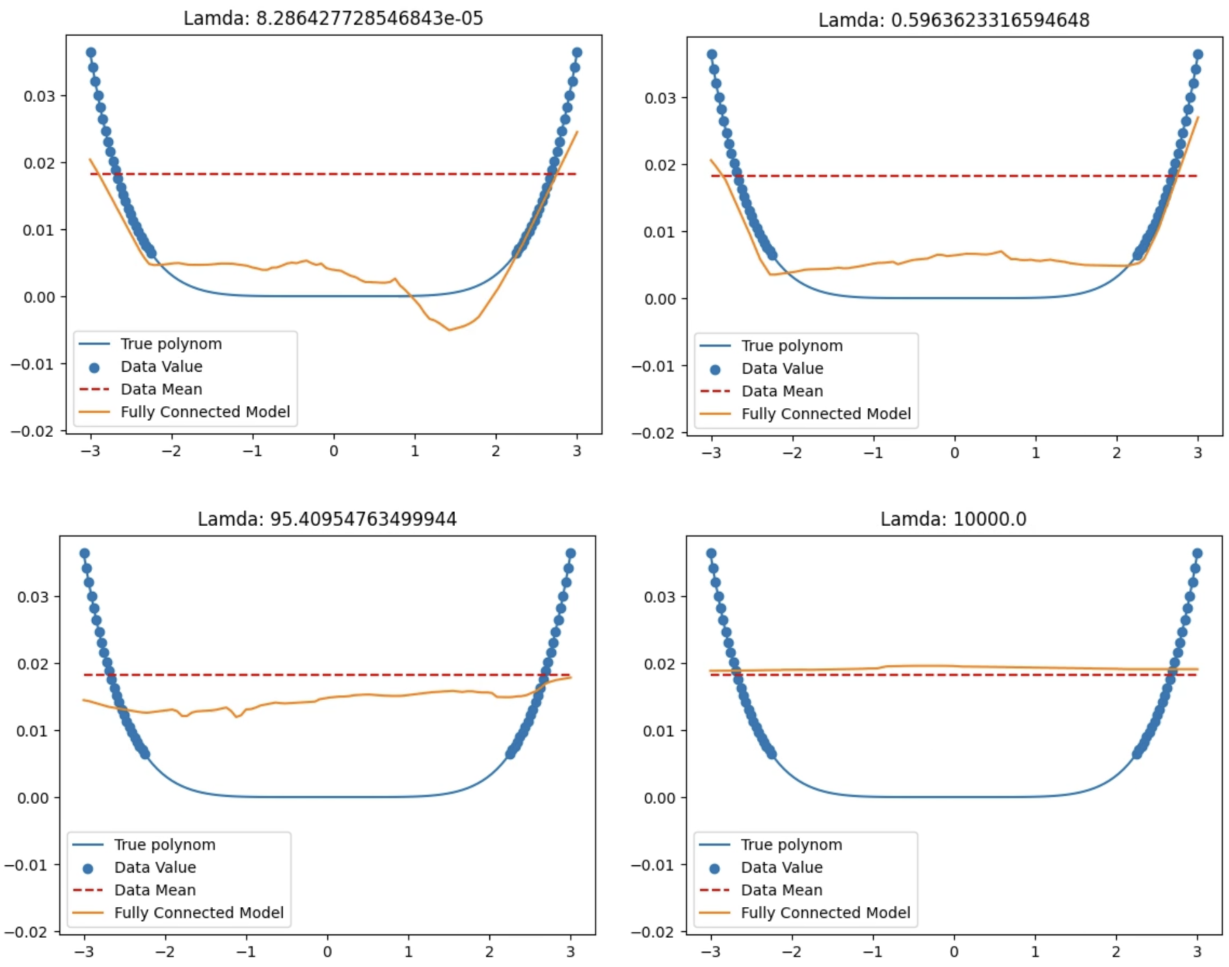


Figure 4: Model evolution for different values of λ (Evolution video : here)

Then, we can state the following Remark 1.

Remark 1. *Our λ must be chosen such that λ is neither $\lambda \ll 1$ nor $\lambda \gg 1$ to ensure that the model does not over-fit the data nor converges to a constant.*

Hence, it becomes interesting to compute our model parameters for different values of λ. We want to check if there exists a λ that would be the best compromise between being too large or too small. Figure 5 shows the evolution curves of train accuracy[6], Lipschitz constant, MSE loss, and the sum of both the Lipschitz loss and the MSE loss.
Using Figure 5 we can conclude that we should indeed choose λ carefully, keeping Remark 1 in mind. But we can't expect to find an all working λ because the choice of such value depends too much on other parameters such as learning rates, optimizer, number of iterations, etc which do not appear in this representation.

2.4 Interesting observed property

As we explained before, the way we choose our data to train a model impacts significantly the results it returns. An edge effect of the Lipschitz regularization given in [Bun22] can worsen these issues. Indeed, we have seen that if there is a data

[6] It should be noted that here the accuracy is computed as follows $\mathcal{A}(f_\theta, \Gamma) = \frac{1}{|\Gamma|}\sum_{(x,y)\in\Gamma} \mathbf{1}_{\{\|y-f_\theta(x)\|\leq\epsilon\}}(x,y)$, or its precision depends on a chosen value of ϵ. In Figure 5 we take $\epsilon = 1e^{-1}$ which explains why the accuracy is always almost 1.

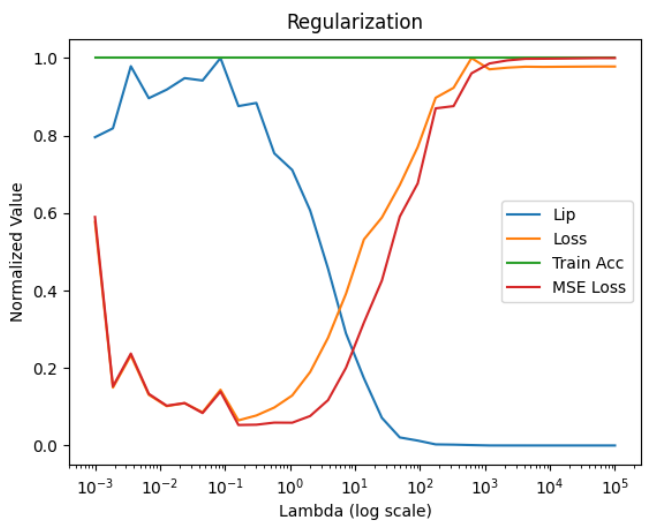


Figure 5: Normalized Lipschitz constant, MSE Loss and train accuracy evolution for different values of λ

loss, this can induce a bias. What is also true is: if for a function $\mathcal{F}$ we want to approximate we have

$$\mathrm{Eq}(\mathcal{F}) = \{(x, \mathcal{F}(x)), x \in \mathcal{X}, \exists\, x' \in \mathcal{X}, \mathcal{F}(x) = \mathcal{F}(x')\} \neq \emptyset$$

The set of point on which $\mathcal{F}$ takes a value it already had taken elsewhere. Because the set $\mathcal{X}_{lip}$ depends on the data, it is possible to have a result that is stuck on the same $\mathcal{X}_{lip}$ for every iteration of CLIP:

Property 1. *Assume we want to approximate a given function $\mathcal{F}$ with the CLIP Algorithm and the data set Γ for a model f_θ. For all $x \in \Gamma \cap \mathbf{Eq}(\mathcal{F})$ there exists $x' \in \mathcal{X}\backslash\{x\}$ such that if $\mathcal{X}^0_{lip} = \{(x, x')\}$ for all $k \in \mathbf{N}$, $\mathcal{X}^k_{lip} = \mathcal{X}^{k+1}_{lip}$.*

Proof. Before proving this we need to detail how $\mathcal{X}_{lip}$ is computed at every iteration. We use SGDM as follows[7] :

$$\text{For all } ({}^kx, {}^kx') \in \mathcal{X}^k_{lip},$$

$$^{k+1}x \leftarrow^k x + \tau\nabla_x(Lip(^kx, ^kx'))^2$$

$$^{k+1}x' \leftarrow^k x' + \tau\nabla_{x'}(Lip(^kx, ^kx'))^2$$

The way we compute the Lipschitz constant approximation is important. Here we would have[8]

$$\mathrm{Lip}(f_\theta, \mathcal{X}^0_{\mathrm{lip}}) = \frac{||f_\theta(x') - f_\theta(x)||}{||x' - x||} = 0$$

Then x (respectively x') is a global minimum thus, a critical point of $\mathrm{Sq}_{lip}(f_\theta, \cdot, x') = (Lip(f_\theta, (\cdot, x')))^2 = \left(\frac{||f_\theta(\cdot) - f_\theta(x')||}{||\cdot - x'||}\right)^2$ (respectively $\mathrm{Sq}_{lip}(f_\theta, x, \cdot) = (Lip(f_\theta, (x, \cdot)))^2 = \left(\frac{||f_\theta(x) - f_\theta(\cdot)||}{||x - \cdot||}\right)^2$) [9]

$$\nabla_x(\mathrm{Lip}(f_\theta, (x, x')))^2 = \nabla_{x'}(\mathrm{Lip}(f_\theta, (x, x')))^2 = 0 \quad \text{thus} \quad \mathcal{X}^{k+1}_{lip} = \mathcal{X}^k_{lip},$$

[7]The gradient computed here is of the function $\mathrm{Sq}_{lip}(f_\theta, x, x')$. This is due to the fact that a differentiable function is required and it still keeps the desired effect of computing the highest value of the Lipschitz constant.

[8]Supposing our model has correctly fitted the data.

[9]Such a function is differentiable, see Property 15 in Appendix.

hence by recurrence resulting in

$$\mathcal{X}_{lip}^0 = \mathcal{X}_{lip}^1 = \mathcal{X}_{lip}^k, \forall k.$$

$\square$

This problem could often arise because when we compute our set $\mathcal{X}_{lip}$, we actually compute each subset made of one element $\{(x, x')\}$ from $\mathcal{X}_{lip}$. And if our set $\mathcal{X}_{lip}$ is stuck at every iteration, we will obviously obtain a poor Lipschitz constant estimation.

But this issue can be solved easily. Indeed, we will now show that this situation never appears if we are to add noise to our data.
To give further context, adding noise to the training data set is often used in form of data augmentation. This added noise helps increase the robustness of our model because it ultimately reduces over-fitting. But in our case, this added noise will also permit us to bypass the issues Property 1 induces.

Let $\mathcal{G}_x$ be a random variable such that $\mathcal{G}_x \sim \mathcal{N}(0, \sigma^2)$. So $\mathcal{G}_x$ takes values in $\mathcal{X}$. We then define $\mathcal{G}_y \sim \mathcal{N}(0, \sigma^2)$ which will take values in $\mathcal{Y}$. Given a training data set Γ, we define $\Gamma_{\mathcal{G}}$ as follows:

$$\Gamma_{\mathcal{G}} = \{(x + \mathcal{G}_x, y + \mathcal{G}_y) \mid (x, y) \in \Gamma\}.$$

Remark 2. *Given the augmented data set $\Gamma_{\mathcal{G}}$ and the original data set Γ, the noise variables $\mathcal{G}_x$ and $\mathcal{G}_y$ ensure that the model f_θ trained on $\Gamma_{\mathcal{G}}$ is robust to over-fitting.*

Explanation : Consider the augmented data set $\Gamma_{\mathcal{G}} = \{(x + \mathcal{G}_x, y + \mathcal{G}_y) \mid (x, y) \in \Gamma\}$. The addition of Gaussian noise $\mathcal{G}_x \sim \mathcal{N}(0, \sigma^2)$ and $\mathcal{G}_y \sim \mathcal{N}(0, \sigma^2)$ ensures that the model f_θ encounters a variety of slightly altered training samples, promoting generalization. The noise introduces variability that helps the model to learn more robust features and reduces the risk of over-fitting to the original data set Γ.

Now, we can detail the following property:

Property 2. *For a non-constant polynomial function $\mathcal{F}$ and for $\mathcal{X}_{lip}^0 \subset \Gamma_{\mathcal{G}}$, we have $\mathbb{P}(\Gamma_{\mathcal{G}} \cap \mathbf{Eq}(\mathcal{F}) \neq \emptyset) = \mathbb{P}(\exists k, \mathcal{X}_{lip}^k \cap \mathbf{Eq}(\mathcal{F}) \neq \emptyset) = 0$. This result still holds even when $\Gamma \cap \mathbf{Eq}(\mathcal{F}) \neq \emptyset$.*

Proof. First, we need to define $\mathbf{Eq}(\mathcal{F}, y) = \{(x, y) \mid x \in \mathcal{X}, \exists x' \in \mathcal{X}, \mathcal{F}(x) = \mathcal{F}(x') = y\}$ called the level set. This then gives us $\mathbf{Eq}(\mathcal{F}) = \bigcup_{y \in \mathcal{Y}} \mathbf{Eq}(\mathcal{F}, y)$. We can easily assume that $\mathbf{Eq}(\mathcal{F}) \neq \emptyset$ because, if not, proving the property is straightforward.

First, we will prove that $\forall y \in \mathcal{Y}$, $\text{Card}(\mathbf{Eq}(\mathcal{F}, y)) \neq +\infty$. Suppose this result is false; then there exists a y for which there exists an infinity of x such that $\mathcal{P}(x) = \mathcal{F}(x) - y = 0$. Since $\mathcal{P}$ is a polynomial function with an infinity of roots, $\mathcal{P} = 0$. This results implies $\mathcal{F}$ being constant and equal to y. As this is absurd, we conclude that $\forall y \in \mathcal{Y}$, $\text{Card}(\mathbf{Eq}(\mathcal{F}, y)) \neq +\infty$.

Since we assumed $\mathbf{Eq}(\mathcal{F}) \neq \emptyset$, there exists a y such that $\text{Card}(\mathbf{Eq}(\mathcal{F}, y)) > 0$. Now we can rewrite :

$$\mathbb{P}(\Gamma_{\mathcal{G}} \cap \mathbf{Eq}(\mathcal{F}) \neq \emptyset) = \mathbb{P}(\exists (x_{\mathcal{G}}, y_{\mathcal{G}}) \in \Gamma_{\mathcal{G}} \cap \mathbf{Eq}(\mathcal{F}))$$

$$= \mathbb{P}(\bigcup_{y \in \mathcal{Y}} (\exists (x_{\mathcal{G}}, y_{\mathcal{G}}) \in \Gamma_{\mathcal{G}} \cap \mathbf{Eq}(\mathcal{F}, y)))$$

$$= \mathbb{P}(\bigcup_{y\in\mathcal{Y}}(\exists(x',y')\in\Gamma,(x'+\mathcal{G}_x,y'+\mathcal{G}_y)\in\mathbf{Eq}(\mathcal{F},y)))$$

$$\leq^{10}\int_{y\in\mathcal{Y}}\mathbb{P}(\exists(x',y')\in\Gamma,\ (x'+\mathcal{G}_x,y'+\mathcal{G}_y)\in\mathbf{Eq}(\mathcal{F},y))dy$$

$$\leq\int_{y\in\mathcal{Y}}\mathbb{P}(\exists(x',y')\in\Gamma,y'+\mathcal{G}_y=y)dy\leq\int_{y\in\mathcal{Y}}\mathbb{P}(\exists(x',y')\in\Gamma,\mathcal{G}_y=y-y')dy$$

Since Γ is the data set, it is finite. Hence $\forall y\in\mathcal{Y}$, $\mathbb{P}(\exists(x',y')\in\Gamma,\mathcal{G}_y=y-y')=0$. This gives us the expected result of $\mathbb{P}(\Gamma_{\mathcal{G}}\cap\mathbf{Eq}(\mathcal{F})\neq\emptyset)=0$. The reasoning is the same for $\mathbb{P}(\exists k,\ \mathcal{X}_{lip}^k\cap\mathbf{Eq}(\mathcal{F})\neq\emptyset)=0$. Indeed, the set $\bigcup_{k\in[|1,N|]}\mathcal{X}_{lip}^k$ is finite, and for the rest, we follow the same reasoning. The process shown so far does not depend on the fact that either $\Gamma\cap\mathbf{Eq}(\mathcal{F})\neq\emptyset$ or $\Gamma\cap\mathbf{Eq}(\mathcal{F})=\emptyset$. □

Property 2 confirms that for our current polynomial approximation problem we do not need to consider issues where the function has repeating values, indeed they happen with a probability 0. We believe that with weaker assumptions on $\mathcal{F}$, it would still be possible to prove this result. However, in our current situation, the given assumptions are sufficient. But these properties clearly indicate that the efficiency of our model can be compromised depending on how the data are handled. Specifically, if we do not add noise or account for missing data, we could induce errors in our model.

3 High Dimensional Experiments

The CLIP algorithm was primarily applied to the MNIST [LeC98] and Fashion-MNIST [Xia17] datasets. These datasets consist of grayscale images with labels. For MNIST, the images are handwritten digits ranging from 0 to 9. Fashion-MNIST, on the other hand, consists of images of clothing items. Although both datasets contain images of low resolution, they still represent high-dimensional matrices. Thus, we have:

$$\mathcal{X}_{\text{MNIST}}=\mathcal{X}_{\text{Fashion-MNIST}}=[0,1]^{28\times28},$$

$$\mathcal{Y}_{\text{MNIST}}=\{\text{"0"},\text{"1"},\text{"2"},...,\text{"9"}\},$$

$$\mathcal{Y}_{\text{Fashion-MNIST}}=\{\text{"0 : T-shirt"},\text{"1 : Trouser"},\text{"2 : Pullover"},...,\text{"9 : Ankle boot"}\}.$$

Our model will take the following form:

$$\mathcal{X}=\mathbb{R}^{784}\rightarrow\text{Image represented as a vector of 28*28 = 784 values.}$$

$$\mathcal{Y}=[0,1]^{10}\rightarrow\text{Return a probability of being the label 0, 1, 2, ..., 9.}$$

Figure 6 and Figure 7 show well-classified and misclassified MNIST images, respectively. The CLIP accuracy in this context is 97%.

3.1 Optimization of CLIP

Firstly, we will attempt to conduct a benchmark of the different optimizers discussed above for CLIP applied to the MNIST dataset.

[10]Here we accept to write such an integral. Indeed even if its value was $+\infty$, the result would be correct assuming that we are in $\overline{\mathbb{R}}$.

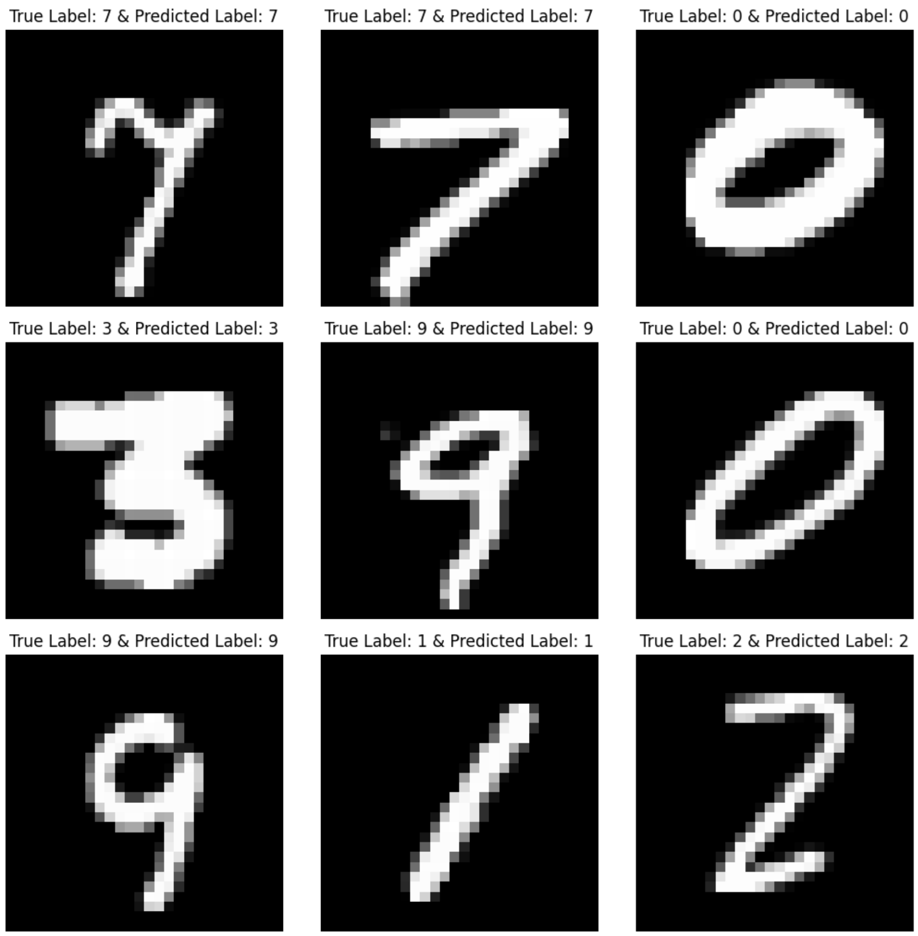

True Label: 7 & Predicted Label: 7
True Label: 7 & Predicted Label: 7
True Label: 0 & Predicted Label: 0
True Label: 3 & Predicted Label: 3
True Label: 9 & Predicted Label: 9
True Label: 0 & Predicted Label: 0
True Label: 9 & Predicted Label: 9
True Label: 1 & Predicted Label: 1
True Label: 2 & Predicted Label: 2

Figure 6: MNIST well classified images

Optimizer	Time (s)	Adv Acc
SGD	299.6679	0.2500
NAG	258.1815	0.2188
ADAM	232.7419	0.2188

Table 1: Performance comparison of different optimizers.

Table 1 shows us the computation time and the accuracy of our model trained with different optimizers (GPU : Nvidia GeForce RTX 3050 Laptop). The accuracy is computed with adversarial attacked data, explaining why the accuracy is this low. We did not show here the basic accuracy because it was sensibly the same for each optimizer. From those results we can conclude that as expected, ADAM is the quickest but loses in accuracy in front of SGD. This time NAG does not seems interesting enough.

3.2 Warm-up Phase

To improve CLIP efficiency, we propose adding a "warm-up" phase to the CLIP algorithm. A warmed-up algorithm typically follows these steps:

- **First Phase:** Train the model on the dataset Γ using only the regular loss (without regularization).
- **Second Phase:** Stop the warm-up phase when the model's training accuracy reaches a predefined threshold α.
- **Third Phase:** Begin training the pre-trained model using the regularizer.

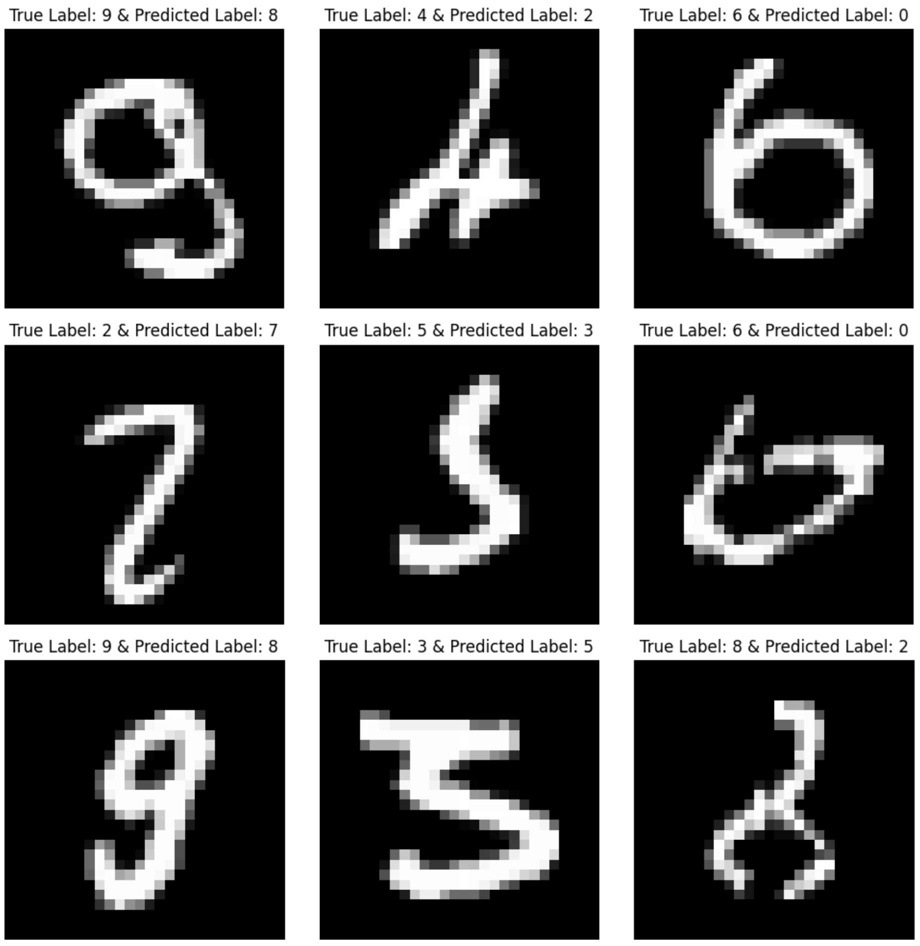

True Label: 9 & Predicted Label: 8
True Label: 4 & Predicted Label: 2
True Label: 6 & Predicted Label: 0
True Label: 2 & Predicted Label: 7
True Label: 5 & Predicted Label: 3
True Label: 6 & Predicted Label: 0
True Label: 9 & Predicted Label: 8
True Label: 3 & Predicted Label: 5
True Label: 8 & Predicted Label: 2

Figure 7: MNIST misclassified image

For our experiments we chose to set the threshold at 0.9. We begin with a standard training for our model. This means we first solve Problem (1). After reaching an accuracy above 0.9 we then begin to regularize with the $CLIP$ algorithm with $ADAM$ optimizer.

4 FLIP - Finite Lipschitz Training

As explained in the introduction, the method of computing the Lipschitz constant is crucial to ensure the effectiveness of our regularization. One such regularization method that indirectly computes the Lipschitz constant locally is Total Variation (TV) Regularization, also known as the ROF model [Rud92]. In this section, we extend the TV regularizer to our Lipschitz Regularization problem based on CLIP, utilizing $\mathcal{X}_{lip}$ and $\text{Lip}(f_\theta, \mathcal{X}_{\text{lip}}) = \sup\limits_{x,x' \in \mathcal{X}_{\text{lip}}} \frac{||f_\theta(x') - f_\theta(x)||}{||x'-x||}$.

4.1 Finite Lipschitz Max and Sum

The algorithm described in [Bun22]: Bungert et al use a max over the $\mathcal{X}_{lip}$ set to estimate the Lipschitz constant. But it would also be interesting to evaluate the mean over the $\mathcal{X}_{lip}$ to take every of those computed constants into account.

Thus in Algorithm 1 we change the regularizer estimator:

$$\textbf{Original Max Estimator : } \mathrm{Lip}(f_\theta, \mathcal{X}_{\mathrm{lip}}) = \sup_{x,x' \in \mathcal{X}_{\mathrm{lip}}} \frac{||f_\theta(x') - f_\theta(x)||}{||x' - x||}$$

$$\textbf{New Sum Estimator : } \mathrm{SumLip}(f_\theta, \mathcal{X}_{\mathrm{lip}}) = \frac{1}{|\mathcal{X}_{lip}|} \sum_{(x,x') \in \mathcal{X}_{\mathrm{lip}}} \frac{||f_\theta(x') - f_\theta(x)||}{||x' - x||}$$

Therefore we now denote the main algorithm - which computes this $\mathcal{X}_{lip}$ set and then regularizes with different estimators upon it - as Finite Lipschitz Algorithm (FLIP)[11]. Then we proceed to a benchmark in order to compare both estimators. In Table 2 we compute the accuracy of Max & Sum estimators on the test dataset with PGD attack ($\epsilon = 0.4$) and FGSM attack ($\epsilon = 0.06$)[12].

Estimator	Test Acc	PGD Adv Acc	FGSM Adv Acc
SUM FLIP	0.9663	0.3584	0.3984
MAX FLIP	0.9685	0.5781	0.5965

Table 2: Performance comparison of Max & Sum Estimator

Even if SUM appears to be less accurate here you will see later with Figure 10, that it does produce interesting results. We remark that while computing this mean estimator, we are approximating a total variation regularizer. We describe below where this observation comes from and what application can be made based on such an approximate TV method.

4.2 Local Lipschitz Constant

First, we define the local Lipschitz constant as follows:

Definition 1. ***Local Lipschitz Constant:*** *For a given* $\epsilon > 0$ *and* $x \in \mathcal{X}$*, we define* $Lip_\epsilon^{loc}(f_\theta, x) = \sup_{x_1,x_2 \in \mathcal{B}_\epsilon(x)} \frac{||f_\theta(x_1) - f_\theta(x_2)||}{||x_1 - x_2||}$ *where* $\mathcal{B}_\epsilon(x) = \{x' \in \mathcal{X},\ \|x' - x\| \leq \epsilon\}$.

Next, we prove some properties of this local Lipschitz constant.

Property 3. $\lim_{\epsilon \to +\infty} Lip_\epsilon^{loc}(f_\theta, x) = Lip(f_\theta, \mathbb{R}^n),\ \forall x \in \mathbb{R}^n$

Proof. Let $x \in \mathbb{R}^n$. Remember that $Lip(f_\theta, \mathbb{R}^n) = \sup_{x,x' \in \mathbb{R}^n} \frac{\|f_\theta(x) - f_\theta(x')\|}{\|x - x'\|}$, thus there exists a sequence $(x_n, x'_n) \in (\mathbb{R}^{2n})^{\mathbb{N}}$ such that

$$\lim_{n \to +\infty} \frac{\|f_\theta(x_n) - f_\theta(x'_n)\|}{\|x_n - x'_n\|} = Lip(f_\theta, \mathbb{R}^n)$$

Now, we have that for every $n \in \mathbb{N}$, there exists $\epsilon_n \geq n > 0$ such that $(x_n, x'_n) \in \mathcal{B}_{\epsilon_n}(x)$.
Indeed, $\epsilon_n = \max(2\|x_n - x\|, 2\|x'_n - x\|, n)$ would works.

$$Lip(f_\theta, \mathbb{R}^n) = \lim_{n \to +\infty} \frac{\|f_\theta(x_n) - f_\theta(x'_n)\|}{\|x_n - x'_n\|} \leq \lim_{n \to +\infty} \sup_{x,x' \in \mathcal{B}_{\epsilon_n}(x)} \frac{\|f_\theta(x) - f_\theta(x')\|}{\|x - x'\|}$$

$$\leq \lim_{\epsilon \to +\infty} \sup_{x,x' \in \mathcal{B}_\epsilon(x)} \frac{\|f_\theta(x) - f_\theta(x')\|}{\|x - x'\|} = \lim_{\epsilon \to +\infty} Lip_\epsilon^{loc}(f_\theta, x)$$

[11]This name has been given in accordance with the original author of [Bun22] which created CLIP Algorithm.
[12]See in Section 7.2 : "Glossary" for an explanation on PGD & FGSM.

Moreover we have that for all $\epsilon > 0$,

$$\sup_{x,x' \in \mathcal{B}_\epsilon(x)} \frac{\|f_\theta(x) - f_\theta(x')\|}{\|x - x'\|} \leq Lip(f_\theta, \mathbb{R}^n)$$

Therefore,

$$\lim_{\epsilon \to +\infty} Lip_\epsilon^{loc}(f_\theta, x) \leq Lip(f_\theta, \mathbb{R}^n)$$

Which finally give us the expected result. □

Remark 3. *A local Lipschitz constant is a Lipschitz constant over a small ball around a given point x. Thus, Property 3 is just a way to say that if we expand this ball indefinitely, then we ultimately obtain the Lipschitz constant over the full set.*

Definition 2. ***Locally Lipschitz Continuous:*** *A function f is locally Lipschitz continuous if and only if for all $x \in \mathcal{X}$, there exists $\epsilon > 0$ such that $Lip_\epsilon^{loc}(f_\theta, x) < +\infty$.*

Property 4. ***(Rademacher Theorem)*** *For a locally Lipschitz continuous function $f : \mathcal{X} \subset \mathbb{R}^n \to \mathbb{R}^m$, f is differentiable almost everywhere.*

Property 5. *For a locally Lipschitz continuous function $f : \mathcal{X} \subset \mathbb{R}^n \to \mathbb{R}$, we have that* ***A.E*** *$x \in \mathcal{X}$, $\lim_{\epsilon \to 0} Lip_\epsilon^{loc}(f, x) = \|\nabla f(x)\|_*$, where $\|\cdot\|_* = \sup_{h \in \mathcal{B}_1(0)} \|\cdot h\|$.*

Proof. Assume we have a function f that is locally Lipschitz continuous. By Property 4, f is differentiable almost everywhere, meaning there exists a set $\tilde{\mathcal{X}}$ such that $\mathbb{L}(\mathcal{X} \backslash \tilde{\mathcal{X}}) = 0$ where $\mathbb{L}$ is the Lebesgue measure[13] and on which f is differentiable[14]. Let $x \in \tilde{\mathcal{X}}$. For all $x' \in \tilde{\mathcal{X}} \cap \mathcal{B}_\epsilon(x) \backslash \{x\}$, we have $|f(x') - f(x)| = |\nabla f(x) \cdot (x' - x)| + o(\|x' - x\|^2)$. Dividing by $\|x' - x\|$ gives

$$\frac{|f(x') - f(x)|}{\|x' - x\|} = \left|\nabla f(x) \cdot \frac{(x' - x)}{\|x' - x\|}\right| + o(\|x' - x\|).$$

Since $\|x' - x\| \leq \epsilon$, we have

$$\frac{|f(x') - f(x)|}{\|x' - x\|} = \left|\nabla f(x) \cdot \frac{(x' - x)}{\|x' - x\|}\right| + o(\epsilon).$$

Taking the supremum,

$$\sup_{x' \in \tilde{\mathcal{X}} \cap \mathcal{B}_\epsilon(x)} \frac{|f(x') - f(x)|}{\|x' - x\|} = \sup_{x' \in \tilde{\mathcal{X}} \cap \mathcal{B}_\epsilon(x)} \left|\nabla f(x) \cdot \frac{(x' - x)}{\|x' - x\|}\right| + o(\epsilon).$$

We know that

$$\sup_{x' \in \tilde{\mathcal{X}} \cap \mathcal{B}_\epsilon(x)} \left|\nabla f(x) \cdot \frac{(x' - x)}{\|x' - x\|}\right| = \sup_{x' \in \tilde{\mathcal{X}} \cap \mathcal{B}_1(x)} |\nabla f(x) \cdot (x' - x)| = \sup_{h \in \mathcal{B}_1(0)} |\nabla f(x) \cdot h| = \|\nabla f(x)\|_*.$$

Thus, we get the expected result:

$$\lim_{\epsilon \to 0} Lip_\epsilon^{loc}(f, x) = \|\nabla f(x)\|_*.$$

□

Remark 4. *We are set in a $\mathbb{R}^n$ space. In such a space, with the usual topology we have an explicit definition of $\|\cdot\|_*$. Indeed for $u \in \mathbb{R}^n$ we get that $\|u\|_* = \|u \cdot \frac{u}{\|u\|}\| = \|u\|$.*

[13] This only means that the set where our function is not differentiable can be overlooked here.
[14] We have a gradient due to Riesz representation theorem in an $\mathbb{R}$-Hilbertian space. [Cia09] : Théorème 1.7. (Représentation de Riesz-Fréchet)

Remark 5. *We can prove such a result with a function $f : \mathcal{X} \subset \mathbb{R}^n \longrightarrow \mathbb{R}^m$. Using that for $x, h \in \mathcal{X}$ with $\|h\| << 1$ we have $f(x+h) = f(x) + \mathcal{J}_f(x) \cdot h + o(\|h\|^2)$ and with a $\mathbb{R}^n$-norm in place of absolute value.*

Without loss of generality we will assume that $\mathcal{Y} \subset \mathbb{R}$. Indeed to extend the result presented below in $\mathcal{Y} \subset \mathbb{R}^m$ we would have to replace gradient by the Jacobian and change norms, which with some more steps would give us the same result.

4.3 Total Variation Regularization

Next, we introduce TV regularization in more detail. TV regularization, detailed in [Rud92] and [Get12], works as follows: For a given set of data Γ, we solve (4):

$$\underset{f_\theta \in \mathbb{BV}(\mathcal{X})}{\operatorname{argmin}} \sum_{(x,y)\in\Gamma} \operatorname{loss}(f_\theta(x), y) + \lambda \|f_\theta\|_{TV} \tag{4}$$

where $\|\cdot\|_{TV} = \int_{\mathcal{X}} \|\nabla \cdot (x)\| \, dx$ is the TV norm and $\mathbb{BV}(\mathcal{X})$ is the space of functions of bounded variation. It is important to note that $\|f_\theta\|_{TV} < +\infty$ for $f_\theta \in \mathbb{BV}(\mathcal{X})$.

The TV regularization has one similar effect as Lipschitz regularization. Indeed, it smooths the model. While Lipschitz regularization provides global smoothing, TV regularization offers local smoothing. This method is often used in image denoising, and the TV norm is primarily based on the Euclidean norm:

$$\|\cdot\|_{TV} = \int_{\mathcal{X}} \|\nabla \cdot (x)\|_2 \, dx.$$

Our contribution here is to suggest using a modified TV regularization for neural network training. As shown previously in Property 5, we have[15]

$$\lim_{\epsilon \to 0} Lip^{loc}_\epsilon(f_\theta, x) = \|\nabla f_\theta(x)\|_*.$$

Hence, a regularization based on total variation can be formulated[16] as follows:

$$\underset{f_\theta \in \mathbb{BV}(\mathcal{X})}{\operatorname{argmin}} \sum_{(x,y)\in\Gamma} \operatorname{loss}(f_\theta(x), y) + \lambda \sum_{x \in \Gamma_{\mathcal{X}}} \|\nabla f_\theta(x)\|_*. \tag{5}$$

This can be interpreted as the following type of regularization:

$$\underset{f_\theta \in \mathbb{BV}(\mathcal{X})}{\operatorname{argmin}} \sum_{(x,y)\in\Gamma} \operatorname{loss}(f_\theta(x), y) + \lambda \int_{\mathcal{X}} \lim_{\epsilon \to 0} Lip^{loc}_\epsilon(f_\theta, x) \, dx. \tag{6}$$

Suppose our model has a Lipschitz constant on the full set of $\mathcal{X}$. We then have that the function $x \mapsto Lip^{loc}_\epsilon(f_\theta, x)$ satisfies:

$$\forall x \in \mathcal{X}, \ |Lip^{loc}_\epsilon(f_\theta, x)| \leq Lip(f_\theta, \mathcal{X}).$$

We have already shown with Property 5, that **A.E.** $x \in \mathcal{X}$, $Lip^{loc}_\epsilon(f_\theta, x) \underset{\epsilon \to 0}{\longrightarrow} \|\nabla f_\theta(x)\|_*$. Let us also suppose that

[15] Indeed our neural network will be considered as locally Lipschitz in futur assumption (Assumption 2).
[16] Considering Property 17 : Norm equivalence in finite dimension.

Algorithm 3: Projected Finite Lipschitz Total Variation

```
for epoch $e = 1$ to $E$ do
    for minibatch $B \subset \Gamma$ do
        $\mathcal{X}_{lip} \leftarrow B$
        for $(x_1, x_2) \in \mathcal{X}_{lip}$ do
            $x \leftarrow x_1$
            $x_1 \leftarrow x_1 + \tau \nabla_{x1} (Lip(x_1, x_2))^2$
            $x_2 \leftarrow x_2 + \tau \nabla_{x2} (Lip(x_1, x_2))^2$
            $x_1 \leftarrow x + \epsilon \frac{(x_1 - x)}{\|x_1 - x\|}$
            $x_2 \leftarrow x + \epsilon \frac{(x_2 - x)}{\|x_2 - x\|}$
        $\theta \leftarrow \mathrm{SGDM}_{\eta,\gamma} \left( \frac{1}{|B|} \sum_{(x,y) \in B} loss(f_\theta(x), y) + \lambda \sum_{(x_1,x_2) \in \mathcal{X}_{lip}} Lip(f_\theta, (x_1, x_2)) \right)$
        $accuracy \leftarrow accuracy + \frac{1}{|B|} \sum_{(x,y) \in B} \mathbf{1}(x, y)_{\{f_\theta(x) = y\}}$
        if $accuracy > \alpha$ then
            $\lambda \leftarrow \lambda + d\lambda$
        else
            $\lambda \leftarrow \lambda - d\lambda$
    $\epsilon \leftarrow \epsilon \times 0.9$
```

$\mathbb{L}(\mathcal{X}) < +\infty$, so that $x \mapsto Lip(f_\theta, \mathcal{X})$, which is a constant function, is integrable[17]

Thus, using Lebesgue's Dominated Convergence Theorem 16, we obtain that:

$$\int_{\mathcal{X}} \lim_{\epsilon \to 0} Lip_\epsilon^{loc}(f_\theta, x)\, dx = \lim_{\epsilon \to 0} \int_{\mathcal{X}} Lip_\epsilon^{loc}(f_\theta, x)\, dx$$

$$= \lim_{\epsilon \to 0} \int_{\mathcal{X}} \sup_{x_1, x_2 \in \mathcal{B}_\epsilon(x)} \frac{||f_\theta(x_1) - f_\theta(x_2)||}{||x_1 - x_2||}\, dx$$

$$= \lim_{\epsilon \to 0} \int_{\mathcal{X}} \sup_{h_1, h_2 \in \mathcal{B}_\epsilon(0)} \frac{||f_\theta(x + h_1) - f_\theta(x + h_2)||}{||h_1 - h_2||}\, dx.$$

4.4 Finite Lipschitz for Total Variation

To implement such a TV regularization, we will partially use the way we computed cheaply a supremum in CLIP, given a $x \in \Gamma_{\mathcal{X}}$ we have :

$$\sup_{h_1, h_2 \in \mathcal{B}_\epsilon(0)} \frac{||f_\theta(x + h_1) - f_\theta(x + h_2)||}{||h_1 - h_2||} \approx \max_{\substack{(x_1, x_2) \in \mathcal{X}_{\text{lip}} \\ s.t\ \|x - x_{1,2}\| \le \epsilon_k}} \frac{||f_\theta(x_1) - f_\theta(x_2)||}{||x_1 - x_2||}$$

$$\forall B \subset \Gamma, \quad pTV_{reg}(f_\theta, B, \epsilon_k) = \sum_{(x,y) \in B} \max_{\substack{(x_1, x_2) \in \mathcal{X}_{\text{lip}} \\ s.t\ \|x - x_{1,2}\| \le \epsilon_k}} \frac{||f_\theta(x_1) - f_\theta(x_2)||}{||x_1 - x_2||} \tag{7}$$

With $\epsilon_k \underset{k \to +\infty}{\longrightarrow} 0$ where k denotes each epoch of our training.
The algorithm we implement then is detailed in Algorithm 3.

[17]Such a hypothesis is not that important, as our dataset is finite, which means we just have to choose our ensemble of origin for our data $\mathcal{X}$ as the largest ball that contains all data (This is later referred as Assumption 3).

Algorithm 4: Random Finite Lipschitz Total Variation

```
for epoch $e = 1$ to $E$ do
    for minibatch $B \subset \Gamma$ do
        $\mathcal{X}_{lip} \leftarrow \emptyset$
        for $x_1 \in B$ do
            $x_2 \leftarrow random\ in\ \mathcal{B}_\epsilon(x_1)$
            $x_1 \leftarrow x_1 + \tau \nabla_{x1}(Lip(x_1, x_2))^2$
            $x_2 \leftarrow x_2 + \tau \nabla_{x2}(Lip(x_1, x_2))^2$
            $\mathcal{X}_{lip} \leftarrow \mathcal{X}_{lip} \cup \{(x_1, x_2)\}$
        $\theta \leftarrow \mathrm{SGDM}_{\eta,\gamma}\left(\frac{1}{|B|}\sum_{(x,y)\in B} loss(f_\theta(x), y) + \lambda \sum_{(x_1,x_2)\in\mathcal{X}_{lip}} Lip(f_\theta, (x_1, x_2))\right)$
        $accuracy \leftarrow accuracy + \frac{1}{|B|}\sum_{(x,y)\in B} \mathbf{1}(x, y)_{\{f_\theta(x)=y\}}$
        if $accuracy > \alpha$ then
            $\lambda \leftarrow \lambda + d\lambda$
        else
            $\lambda \leftarrow \lambda - d\lambda$
    $\epsilon \leftarrow \epsilon \times 0.9$
```

As $\mathcal{X}_{lip}$ is generated by local method we can also implement a "random" TV algorithm without projection. See Algorithm 4 for a detail of this "random" TV.

4.5 Existence of Solution for TV FLIP

We will need to make similar assumptions as the one made in CLIP [Bun22].

Here we first detail those three assumptions:

Assumption 1. *We will suppose that our loss function verifies:*

- *1. That $loss(y_1, y_2) \geq 0,\ \forall y_1, y_2 \in \mathcal{Y}$.*
- *2. That $loss(\cdot, y)$ is a lower semi-continuous[18] and convex[19] function for all $y \in \mathcal{Y}$.*
- *3. That there exist $\theta \in \Theta$ such that $\exists (x_0, y_0) \in \Gamma,\ loss(f_\theta(x_0), y_0) \in \mathbb{R}$ and $\forall (x, y) \in \Gamma,\ loss(f_\theta(x_0), y_0) \neq +\infty$.(Our loss function is proper)[20]*

Assumption 2. *Let suppose that our neural networks f_θ satisfy:*

- *1. That $x \mapsto f_\theta(x)$ is Lipschitz continuous. Thus it is also locally Lipschitz continuous.[21]*
- *2. That $\theta \mapsto f_\theta(x)$ is continuous for all $x \in \mathcal{X}$.*
- *3. That there exists $\theta \in \Theta$ such that f_θ is constant.*

[18] A function $f : \mathcal{X} \longrightarrow \mathcal{Y}$ is lower semi-continuous if and only if $\liminf_{x\to x_0} f(x) \geq f(x_0), \forall x_0 \in \mathcal{X}$.

[19] Cross-Entropy loss and Mean Squared loss are convex functions for example. A convex function $f : \mathcal{X} \to \mathcal{Y}$ is a function such that $\forall x, x' \in \mathcal{X}$, $\forall t \in [0, 1]$, $f(tx + (1-t)x') \leq tf(x) + (1-t)f(x)$.

[20] This is obviously the case for many commonly used loss functions that can be computed.

[21] Using Rademacher Theorem 4 we even get that f_θ is differentiable almost everywhere.

Assumption 3. *We have that there exists $n \in \mathbb{N}$ such that the set $\mathcal{X} \subset \mathbb{R}^n$. Let suppose that the set $\mathcal{X}$ is bounded, open and connected (We could imagine for example $\mathcal{X} = \{x \in \mathbb{R}^n,\ \|x\| < 1 + \underset{x_\gamma \in \Gamma_\mathcal{X}}{sup} \|x_\gamma\|\}$).*

Using these assumptions, we show that:

Property 6. ***(Finite problem)*** *Under Assumption 1-3, there exists $\theta \in \Theta$ such that*

$$\sum_{(x,y)\in\Gamma} loss(f_\theta(x), y) + \lambda \int_\mathcal{X} \lim_{\epsilon\to 0} Lip_\epsilon^{loc}(f_\theta, x)\, dx < +\infty.$$

Proof. First, we aim to prove that

$$\int_\mathcal{X} \lim_{\epsilon\to 0} Lip_\epsilon^{loc}(f_\theta, x)\, dx < +\infty.$$

Thanks to Assumptions 2 and 3, we have

$$\lim_{\epsilon\to 0} Lip_\epsilon^{loc}(f_\theta, x) \leq Lip(f_\theta, \mathcal{X})$$

and

$$\int_\mathcal{X} Lip(f_\theta, \mathcal{X})\, dx = Lip(f_\theta, \mathcal{X})\mathbb{L}(\mathcal{X}) < +\infty.$$

Thus, we have

$$\int_\mathcal{X} \lim_{\epsilon\to 0} Lip_\epsilon^{loc}(f_\theta, x)\, dx \leq \int_\mathcal{X} Lip(f_\theta, \mathcal{X})\, dx < +\infty.$$

Next, let us show that there exists $\theta \in \Theta$ such that

$$\sum_{(x,y)\in\Gamma} \text{loss}(f_\theta(x), y) < +\infty.$$

First, note that Γ is our dataset, so we can consider it finite. Therefore, we need to prove that there exists $\theta \in \Theta$ such that

$$\forall (x,y) \in \Gamma,\ \text{loss}(f_\theta(x), y) < +\infty.$$

Using Assumption 1, we know that our loss function is convex and lower semi-continuous. Moreover, there exists $\theta \in \Theta$ such that there is at least one point in our data where the loss is finite and no point for which it is equal to $+\infty$. Consequently, the loss function is finite for every point in this given case[22]. Thus, we have proved the expected property[23]. □

Lemma 1. *Under Assumption 2, we have that $\theta \mapsto \int_\mathcal{X} \lim_{\epsilon\to 0} Lip_\epsilon^{loc}(f_\theta, x)\, dx$ is lower semi-continuous.*

Proof. First of all, using Assumption 2, we have that $\theta \mapsto f_\theta$ is continuous. To prove our result we will use the following form of TV regularizer[24]:

$$\int_\mathcal{X} \lim_{\epsilon\to 0} Lip_\epsilon^{loc}(f_{\theta_0}, x)\, dx = \int_\mathcal{X} \|\nabla f_\theta(x)\|\, dx$$

It was proven here [Cha09] that such a formula is lower semi-continuous upon the function it is applied on -in our case f_θ- and thanks to $\theta \mapsto f_\theta$ being continuous we get that our current regularizer term is lower semi-continuous upon θ.

[22]Property 18 gives us that our *loss* function won't take $-\infty$ as value in our case.
[23]It would have been possible to prove this also using f_{θ_c} a constant model but as we shown above this assumption is not needed for this result
[24]Using Remark 4, we have that here $\|\nabla f_\theta(x)\|_* = \|\nabla f_\theta(x)\|,\ \forall x \in \mathcal{X}$.

With greater assumption such as continuity of $\theta \mapsto \nabla f_\theta(x)$ would directly gives us the expected result using Property 16 and norm continuity. This assumption would be correct for most neural networks structures but not all. $\square$

Property 7. ***(TV Problem Solution Existence)*** *For a closed bounded subset $\Theta_B \subset \Theta$ with Assumptions 1-3, we have that for all $\lambda > 0$, Problem (6) admits at least one solution in Θ_B.*

Proof. The structure of this proof is also similar to the existence proof given in CLIP [Bun22].

First, let us take some closed bounded subset $\Theta_b \subset \Theta$ and denote $\theta_0 \in \Theta$ as the coefficient which verifies Assumption 1.3. We set $\Theta_B = \Theta_b \cup \{\theta_0\}$, which is still a closed and bounded subset. Using Assumptions 1-3, we have Property 6, which allows us to define $(\theta_i)_{i\in\mathbb{N}} \in \Theta_B^{\mathbb{N}}$ as a minimizing sequence for Problem (6). Recall that Θ is a normed $\mathbb{R}$-vectorial space of finite dimension[25]. Using the Bolzano-Weierstrass Theorem (Property 19), $(\theta_i)_{i\in\mathbb{N}}$ admits a convergent subsequence $(\theta_{\phi(i)})_{i\in\mathbb{N}}$ with limit θ^*. Since Θ_B is closed, $\theta^* \in \Theta_B$. Using Lemma 1, Assumption 1.2, and Assumption 2, we have that $\theta \mapsto \sum_{(x,y)\in\Gamma} \mathrm{loss}(f_\theta(x), y) + \lambda \int_{\mathcal{X}} \lim\limits_{\epsilon\to 0} Lip_\epsilon^{loc}(f_\theta, x)\, dx$ is lower semi-continuous. Hence, θ^* is a solution of Problem (6). $\square$

Remark 6. *The result from Property 7 still holds for a problem with ϵ fixed (not a limit).*
i.e *: For a closed bounded subset $\Theta_B \subset \Theta$ with Assumptions 1-3, we have that Problem (8) :*

$$\min_{\theta\in\Theta} \sum_{(x,y)\in\Gamma} loss(f_\theta(x), y) + \lambda \int_{\mathcal{X}} Lip_\epsilon^{loc}(f_\theta, x)\, dx \tag{8}$$

admits at least one solution in Θ_B. Indeed, we won't show it here but we have the lower semi-continuity of $\theta \mapsto \int_{\mathcal{X}} Lip_\epsilon^{loc}(f_\theta, x)\, dx$, thus we can reuse the proof structure.

4.6 Properties for TV FLIP

For this subsection we will suppose that Assumptions 1-3 are always verified.

4.6.1 Infinite lambda

Firstly, we want to prove that for $\lambda \longrightarrow +\infty$ we have a similar result as the one from CLIP. We expect to get a constant model when we over-regularize with high λ.

Property 8. *Under Assumptions 2 and 3, we have that our model $f_\theta \in \mathcal{H}^{1,2}(\mathcal{X})$. Where $\mathcal{H}^{1,2}(\mathcal{X})$ is a Sobolev space*[26].

Proof. Using Assumption 2 and Rademacher Theorem 4, we first show that f_θ is differentiable almost everywhere. This assumption also implies that f_θ is continuous everywhere which makes it measurable. Then, Property 5 and the continuity of $x \mapsto x^2$ gives us :

$$\textbf{A.E}\ x \in \mathcal{X},\ \|\nabla f_\theta(x)\|_*^2 = \lim_{\epsilon\to 0} \left(Lip_\epsilon^{loc}(f_\theta, x)\right)^2$$

And Assumption 2 finally yields:

$$\|\nabla f_\theta(x)\|_*^2 \leq \left(Lip(f_\theta, \mathcal{X})\right)^2$$

[25] For a neural network with $k \in \mathbb{N}$ layers of $n_0, n_1, \ldots, n_k \in \mathbb{N}$ weights, we have $\Theta = \mathbb{R}^{n_0+n_1+\cdots+n_k}$.
[26] See Notation 14.

Thus, with Assumption 3 we get :

$$\int_{\mathcal{X}} \|\nabla f_\theta(x)\|_*^2 \, dx \ \leq \ \int_{\mathcal{X}} \left(Lip(f_\theta, \mathcal{X})\right)^2 \, dx \leq \left(Lip(f_\theta, \mathcal{X})\right)^2 \mathbb{L}(\mathcal{X}) < +\infty$$

Finally, thanks to Property 17, we have the norm equivalence in a finite dimensional space. Thus $\nabla f_\theta \in L^2(\mathcal{X})$ which results in $f_\theta \in \mathcal{H}^{1,2}(\mathcal{X})$. $\square$

Property 9. *Under Assumption 2, there exists θ_∞^* a limit : $\theta_{\lambda_n}^* \underset{\lambda_n \to +\infty}{\longrightarrow} \theta_\infty^*$ such that Problem (6) with λ_n is solved by $f_{\theta_{\lambda_n}^*}$ and $\underset{\lambda_n \to +\infty}{lim} \sum_{(x,y)\in\Gamma} loss(f_{\theta_{\lambda_n}^*}(x), y) + \lambda_n \int_{\mathcal{X}} \lim_{\epsilon\to 0} Lip_\epsilon^{loc}(f_{\theta_{\lambda_n}^*}, x)\, dx < +\infty$ for Problem (6) when $\lambda_n \longrightarrow +\infty$.*

Proof. Under Assumption 2.3 we have that there exists $\theta_c \in \Theta$ such that f_{θ_c} is constant. Let denote $M = \sum_{(x,y)\in\Gamma} \text{loss}(f_{\theta_c}(x), y)$. We have that for all λ:

$$\sum_{(x,y)\in\Gamma} \text{loss}(f_{\theta_c}(x), y) + \lambda \int_{\mathcal{X}} \lim_{\epsilon\to 0} Lip_\epsilon^{loc}(f_{\theta_c}, x)\, dx =^{27} \sum_{(x,y)\in\Gamma} \text{loss}(f_{\theta_c}(x), y)$$

$$= M$$

This in fact means that for all sequence $(\lambda_n)_{n\in\mathbb{N}}$, we have that $\left(\sum_{(x,y)\in\Gamma} \text{loss}(f_{\theta_{\lambda_n}^*}(x), y) + \lambda_n \int_{\mathcal{X}} \lim_{\epsilon\to 0} Lip_\epsilon^{loc}(f_{\theta_{\lambda_n}^*}, x)\, dx\right)_{n\in\mathbb{N}}$ is bounded by M. Indeed, if we assume that it is not the case then there would be a N such that :

$$M < \sum_{(x,y)\in\Gamma} \text{loss}(f_{\theta_{\lambda_N}^*}(x), y) + \lambda_N \int_{\mathcal{X}} \lim_{\epsilon\to 0} Lip_\epsilon^{loc}(f_{\theta_{\lambda_N}^*}, x)\, dx$$

Which results in θ_c being a strictly better minimizer for Problem (6) with such a λ_N. Thus it is not possible because $\theta_{\lambda_N}^*$ was defined as a minimal solution.

Then if we choose a sequence $(\lambda_n)_{n\in\mathbb{N}}$ such that $\lambda_n \underset{n\to+\infty}{\longrightarrow} +\infty$ (we can easily choose for example $\lambda_n = n$). Therefore as we have that $\left(\sum_{(x,y)\in\Gamma} \text{loss}(f_{\theta_{\lambda_n}^*}(x), y) + \lambda_n \int_{\mathcal{X}} \lim_{\epsilon\to 0} Lip_\epsilon^{loc}(f_{\theta_{\lambda_n}^*}, x)\, dx\right)_{n\in\mathbb{N}}$ is bounded (by 0 and M), using Bolzano-Weierstrass Theorem 19 we have that there exists :

$$\phi : \mathbb{N} \longrightarrow \mathbb{N}, \text{ strictly increasing, such that}$$

$$\left(\sum_{(x,y)\in\Gamma} \text{loss}(f_{\theta_{\lambda_{\phi(n)}}^*}(x), y) + \lambda_{\phi(n)} \int_{\mathcal{X}} \lim_{\epsilon\to 0} Lip_\epsilon^{loc}(f_{\theta_{\lambda_{\phi(n)}}^*}, x)\, dx\right)_{n\in\mathbb{N}} \text{ converges towards } M_\infty < +\infty$$

In consideration that we search $\theta_\infty^* \in \overline{\Theta} = \overline{\mathbb{R}}^p = (\mathbb{R} \cup \{-\infty, +\infty\})^p$ we can observe that if our sequence $(\theta_{\lambda_n}^*)_{n\in\mathbb{N}}$ is bounded then we have Bolzano-Weierstrass, if not we have a subsequential limit where every coordinate tends to a value in $\overline{\mathbb{R}}$. Which means there exists a function $\psi : \mathbb{N} \longrightarrow \mathbb{N}$, strictly increasing, such that $\theta_{\lambda_{\phi\circ\psi(n)}}^* \underset{n\to+\infty}{\longrightarrow} \theta_\infty^* \in \overline{\Theta}$ and $\underset{n\to+\infty}{lim} \sum_{(x,y)\in\Gamma} \text{loss}(f_{\theta_{\lambda_{\phi\circ\psi(n)}}^*}(x), y) + \lambda_{\phi\circ\psi(n)} \int_{\mathcal{X}} \lim_{\epsilon\to 0} Lip_\epsilon^{loc}(f_{\theta_{\lambda_{\phi\circ\psi(n)}}^*}, x)\, dx < +\infty$. $\square$

Property 10. ***(Infinite Lambda)*** *: Under Assumptions 2 & 3 - we will denote θ_∞^* the same limit seen in Property 9 - we have that such a θ_∞^* verifies that $f_{\theta_\infty^*}$ is constant.*

Proof. Under Assumption 2, we have from Property 9 that there is a limit θ_∞^* which verifies that there exists a sequence

[27] Because $\forall x \in \mathcal{X},\ Lip_\epsilon^{loc}(f_{\theta_c}, x) = 0$.

of solutions verifying $\theta^*_\lambda \underset{\lambda\to+\infty}{\longrightarrow} \theta^*_\infty$ and $\lim\limits_{\lambda\to+\infty}\left(\sum_{(x,y)\in\Gamma}\text{loss}(f_{\theta^*_\lambda}(x),y)+\lambda\int_{\mathcal{X}}\lim\limits_{\epsilon\to 0} Lip^{loc}_\epsilon(f_{\theta^*_\lambda},x)\,dx\right)<+\infty$.
Both $\sum_{(x,y)\in\Gamma}\text{loss}(f_{\theta^*_\lambda}(x),y)$ and $\lambda\int_{\mathcal{X}}\lim\limits_{\epsilon\to 0} Lip^{loc}_\epsilon(f_{\theta^*_\lambda},x)\,dx$ are positive terms, and thus both parts of the sum are finite.

Thus, we have that $\lim\limits_{\lambda\to+\infty}\lambda\int_{\mathcal{X}}\lim\limits_{\epsilon\to 0} Lip^{loc}_\epsilon(f_{\theta^*_\lambda},x)\,dx<+\infty$, which means that $\lim\limits_{\lambda\to+\infty}\int_{\mathcal{X}}\lim\limits_{\epsilon\to 0} Lip^{loc}_\epsilon(f_{\theta^*_\lambda},x)\,dx=0$. By Lemma 1, we have that $\int_{\mathcal{X}}\lim\limits_{\epsilon\to 0} Lip^{loc}_\epsilon(f_{\theta^*_\infty},x)\,dx=0$, which gives us that $f_{\theta^*_\infty}$ is locally Lipschitz [28]. Thus using Property 5 this integral can be rewritten as $\int_{\mathcal{X}}\|\nabla f_{\theta^*_\infty}(x)\|_*\,dx=0$. Then, we get that **A.E.** $x\in\mathcal{X},\ \|\nabla f_{\theta^*_\infty}(x)\|_*=0$, giving us **A.E.** $x\in\mathcal{X},\ \nabla f_{\theta^*_\infty}(x)=0$.

Using that $\int_{\mathcal{X}}\|\nabla f_{\theta^*_\infty}(x)\|_*\,dx=0$ and that $f_{\theta^*_\infty}$ is locally Lipschitz (thus continuous and differentiable almost everywhere) we can prove similarly as Property 8 Proof that $f_{\theta^*_\infty}\in\mathcal{H}^{1,1}(\mathcal{X})$. Then because $\mathcal{X}$ is a connected domain[29] and $f_{\theta^*_\infty}\in\mathcal{H}^{1,1}(\mathcal{X})$, Property 20 gives us that almost everywhere $f_{\theta^*_\infty}$ is indeed constant. Finally, with continuity, we then obtain that $f_{\theta^*_\infty}$ is constant on the whole $\mathcal{X}$ set. □

This property confirms an empirically observed result: our model flattens when its regularizer's weight increases. Now, we will give a more accurate result on this constant function.

Property 11. *Under the conditions of Property 10, we have that* $f_{\theta^*_\infty}=\arg\min\limits_{y_\theta\in\mathcal{Y}_\Theta}\left\{\sum_{(x,y)\in\Gamma}loss(y_\theta,y)\right\}$, *where* $\mathcal{Y}_\Theta=\{y\in\mathcal{Y}\mid\exists\theta\in\Theta,\ \forall x\in\mathcal{X},\ f_\theta(x)=y\}$.

Proof. (This part of the proof is again similar to the one given in CLIP [Bun22]) Under the conditions of Property 10, $f_{\theta^*_\infty}$ is constant. Hence, there exists $\tilde{y}\in\mathcal{Y}$ such that $\forall x\in\mathcal{X},\ f_{\theta^*_\infty}(x)=\tilde{y}$. Using the fact that our sequence $(\theta^*_\lambda)_\lambda$ is an optimal sequence for every λ, we obtain that:

$$\sum_{(x,y)\in\Gamma}\text{loss}(f_{\theta^*_\lambda}(x),y)+\lambda\int_{\mathcal{X}}\lim_{\epsilon\to 0} Lip^{loc}_\epsilon(f_{\theta^*_\lambda},x)\,dx\le\sum_{(x,y)\in\Gamma}\text{loss}(f_\theta(x),y)+\lambda\int_{\mathcal{X}}\lim_{\epsilon\to 0} Lip^{loc}_\epsilon(f_\theta,x)\,dx.$$

This even holds for functions f_θ that are constant. We denote them y_θ and confound their value in $\mathcal{Y}$ with the function itself:

$$\sum_{(x,y)\in\Gamma}\text{loss}(f_{\theta^*_\lambda}(x),y)+\lambda\int_{\mathcal{X}}\lim_{\epsilon\to 0} Lip^{loc}_\epsilon(f_{\theta^*_\lambda},x)\,dx\le\sum_{(x,y)\in\Gamma}\text{loss}(y_\theta,y)+\lambda\int_{\mathcal{X}}\lim_{\epsilon\to 0} Lip^{loc}_\epsilon(y_\theta,x)\,dx.$$

But for such constant functions, we have that:

$$Lip^{loc}_\epsilon(y_\theta,x)=0,\ \forall x\in\mathcal{X},\ \forall\epsilon>0.$$

Thus:

$$\sum_{(x,y)\in\Gamma}\text{loss}(f_{\theta^*_\lambda}(x),y)+\lambda\int_{\mathcal{X}}\lim_{\epsilon\to 0} Lip^{loc}_\epsilon(f_{\theta^*_\lambda},x)\,dx\le\sum_{(x,y)\in\Gamma}\text{loss}(y_\theta,y).$$

Using Lemma 1 and Assumption 1.2, we have the lower semi-continuity of our right term, which gives us:

$$\sum_{(x,y)\in\Gamma}\text{loss}(f_{\theta^*_\infty}(x),y)+\lambda\int_{\mathcal{X}}\lim_{\epsilon\to 0} Lip^{loc}_\epsilon(f_{\theta^*_\infty},x)\,dx\le\liminf_{\lambda\to+\infty}\sum_{(x,y)\in\Gamma}\text{loss}(f_{\theta^*_\lambda}(x),y)+\lambda\int_{\mathcal{X}}\lim_{\epsilon\to 0} Lip^{loc}_\epsilon(f_{\theta^*_\lambda},x)\,dx$$

$$\le\sum_{(x,y)\in\Gamma}\text{loss}(y_\theta,y).$$

[28]We cannot use Assumption 2 for $f_{\theta^*_\infty}$: Because it is only a limit of function verifying Assumption 2 but the Lipschitz constant could explode even if it is locally stable. This even holds when the integral of a local Lipschitz constant is finite, indeed, we could have a subset of measure 0 on which the Lipschitz constant explodes.
[29]With Assumption 3, $\mathcal{X}$ is indeed an open and connected set.

As we have that $\int_{\mathcal{X}} \lim_{\epsilon \to 0} Lip_{\epsilon}^{loc}(f_{\theta^*_\infty}, x)\, dx = 0$, we obtain:

$$\sum_{(x,y)\in\Gamma} \text{loss}(\tilde{y}, y) \leq \sum_{(x,y)\in\Gamma} \text{loss}(y_\theta, y).$$

This proves our property. □

Remark 7. *As Bungert et al. remarked in [Bun22], if our loss function is the Euclidean distance $loss(y, y') = \|y - y'\|_2^2$ (our norm is the Euclidean norm), we have that $f_{\theta^*_\infty}$ is a projection of our training sample $\Gamma_{\mathcal{Y}}$ barycenter on the $\mathcal{Y}_\Theta$ set.*

Remark 8. *In CLIP [Bun22], Proposition 3 gives us that under the assumption that $\min_{\theta\in\Theta} \frac{1}{|\Gamma|} \sum_{(x,y)\in\Gamma} loss(f_\theta(x), y) = 0$ we can give a characterization of a possible limit when $\lambda \longrightarrow 0$. Using a same reasoning we would get a similar result. This proves that when $\lambda \longrightarrow 0$, the model obtained (if it does converge) is the one which minimizes the most our regularizer term while perfectly fitting the data relatively to the loss function definition.*

4.6.2 Gamma convergence of the Lipschitz learning for TV

Given $(\epsilon_n)_n \in (\mathbb{R}_+^*)^{\mathbb{N}}$, Algorithms 3 and 4 both search a solution $\theta^*_{\epsilon_n}$ of the following problem :

$$\min_{\theta\in\Theta} \sum_{(x,y)\in\Gamma} \text{loss}(f_\theta(x), y) + \lambda \int_{\mathcal{X}} Lip_{\epsilon_n}^{loc}(f_\theta, x)\, dx. \tag{9}$$

At every epoch n, with $\epsilon_n \underset{n\to+\infty}{\longrightarrow} 0$.
We could get the limit over ϵ out of the integral using Lebesgue's dominated convergence theorem, however we need to confirm that $f_{\theta^*_{\epsilon_n}} \underset{n\to+\infty}{\longrightarrow} f_{\theta^*_{TV}}$ with $f_{\theta^*_{TV}}$ a solution for Problem (6) or with some weaker convergence.
We are going to use the similar structure of proof given in [Shi24] (page 11-14) to prove the weaker γ-convergence of our regularizer.

Definition 3. ***(Gamma convergence)*** *Let $(f_n)_{n\in\mathbb{N}} \in \mathcal{F}(\Omega, \overline{\mathbb{R}})^{\mathbb{N}}$ and $f \in \mathcal{F}(\Omega, \overline{\mathbb{R}})$. We have that $f_n \xrightarrow{\gamma} f$ if and only if $\forall x \in \Omega$:*

- ***Liminf inequality*** *: $\forall (x_n)_{n\in\mathbb{N}} \in \Omega^{\mathbb{N}},\ x_n \underset{n\to+\infty}{\longrightarrow} x, \quad \liminf_{n\to\infty} f_n(x_n) \geq f(x)$*
- ***Limsup inequality*** *: $\exists (x_n)_{n\in\mathbb{N}} \in \Omega^{\mathbb{N}},\ x_n \underset{n\to+\infty}{\longrightarrow} x, \quad \limsup_{n\to\infty} f_n(x_n) \leq f(x)$*

Property 12. *Let $(f_n)_{n\in\mathbb{N}} \in \mathcal{F}(\Omega, \overline{\mathbb{R}})^{\mathbb{N}}$ and $f \in \mathcal{F}(\Omega, \overline{\mathbb{R}})$ with $f_n \underset{n\to+\infty}{\overset{\gamma}{\longrightarrow}} f$.
If there exists a pre-compact sequence $\{x_n\}_{n\in\mathbb{N}}$ such that*

$$\lim_{n\to\infty} \left(f_n(x_n) - \inf_{x\in\Omega} f_n(x) \right) = 0,$$

then

$$\lim_{n\to\infty} \inf_{x\in\Omega} f_n(x) = \inf_{x\in\Omega} f(x),$$

and all subsequential limits of $\{x_n\}_{n\in\mathbb{N}}$ is a minimizer of f.

Now :

Let $p > 1$.

Let set $\Omega = L^p(\mathcal{X})$. Then let us define for $u \in L^p(\mathcal{X})$

$$\mathcal{E}_\varepsilon(u) = \int_{\mathcal{X}} \underset{z,z' \in \mathcal{B}_\varepsilon(x)}{\text{ess sup}} \frac{|u(z)-u(z')|^p}{|z-z'|^p} dx,$$

We want to prove the γ-convergence of $\mathcal{E}_\varepsilon$ to

$$\mathcal{E}(u) = \begin{cases} \int_{\mathcal{X}} |\nabla u|^p dx, & \text{if } u \in \mathcal{H}^{1,p}(\mathcal{X}), \\ +\infty, & \text{otherwise,} \end{cases}$$

in $L^p(\mathcal{X})$ as $\varepsilon \to 0$.

Lemma 2. *Let $\{u_\varepsilon\}$ be a sequence of uniformly bounded $C^2(\mathcal{X})$ functions. If $\nabla u_\varepsilon \to \nabla u$ in $L^p(\mathcal{X})$ as $\varepsilon \to 0$, then*

$$\lim_{\varepsilon \to 0} \mathcal{E}_\varepsilon(u_\varepsilon) = \mathcal{E}(u).$$

Proof. See Lemma 3.3 Proof in [Shi24]. □

Using Lemma 2 we can show :

Lemma 3. *If $u_\varepsilon \to u$ in $L^p(\mathcal{X})$ as $\varepsilon \to 0$, then*

$$\liminf_{\varepsilon \to 0} \mathcal{E}_\varepsilon(u_\varepsilon) \geq \mathcal{E}(u).$$

Proof. See Lemma 3.4 Proof in [Shi24]. □

Lemma 4. *For any $u \in L^p(\mathcal{X})$, there exists a sequence of functions $\{u_\varepsilon\} \subset L^p(\mathcal{X})$ such that $u_\varepsilon \to u$ in $L^p(\mathcal{X})$ as $\varepsilon \to 0$ and*

$$\limsup_{\varepsilon \to 0} \mathcal{E}_\varepsilon(u_\varepsilon) \leq \mathcal{E}(u).$$

Proof. See Lemma 3.5 Proof in [Shi24]. □

Finally, using Lemma 3 and 4 we obtain the expected result :

Property 13. *Let $1 < p < \infty$. Then*

$$\mathcal{E}_\varepsilon \xrightarrow{\gamma} \mathcal{E},$$

in $L^p(\mathcal{X})$ as $\varepsilon \to 0$.

Proof. See Theorem 3.6 in [Shi24]. □

Thus, using Property 12 and that $f \mapsto \sum_{(x,y)\in\Gamma} loss(f(x), y)$ is independent of ε we get :

Property 14. *If $p > n$[30] and $\Omega_B \subset \mathcal{H}^{1,p}(\mathcal{X})$ is bounded, we have that forall $(f_k^*)_k \in \Omega_B^{\mathbb{N}}$, such that f_k^* solves $\underset{f \in \Omega_B}{inf} \sum_{(x,y)\in\Gamma} loss(f(x), y) + \lambda \mathcal{E}_\varepsilon(f)$, then every subsequential limit f^* of $(f_k^*)_k$ solves $\underset{f \in \Omega_B}{inf} \sum_{(x,y)\in\Gamma} loss(f(x), y) + \lambda \mathcal{E}(f)$.*

[30] We have that n is the integer for which $\mathcal{X} \subset \mathbb{R}^n$. As we are in high dimension we have that n is important, for example with MNIST $n = 784$.

Proof. Firstly, let us define $p > n$. Using Property 23 (Sobolev Embedding Theorem) we obtain that $\mathcal{H}^{1,p}(\mathcal{X}) \subset C^0(\mathcal{X})$. Thus, for $u \in \mathcal{H}^{1,p}(\mathcal{X})$, we are allowed[31] to write :

$$\mathcal{K}_\varepsilon(u) = \sum_{(x,y)\in\Gamma} loss(u(x), y) + \lambda\mathcal{E}_\varepsilon(u)$$

$$\mathcal{K}(u) = \sum_{(x,y)\in\Gamma} loss(u(x), y) + \lambda\mathcal{E}(u)$$

Then, using Remark 6 we obtain our sequences $(f_k^*)_k \in \Omega_B^{\mathbb{N}}$ which do have subsequences converging to different limits f^* (using Property 19, Bolzano-Weierstrass). This sequence is in a bounded subset Ω_B, thus is pre-compact and verifies that $\mathcal{K}_\varepsilon(f_k^*) = \underset{f\in\Omega_B}{inf}\,\mathcal{K}_\varepsilon(f)$. Moreover as $f \mapsto \sum_{(x,y)\in\Gamma} loss(f(x), y)$ is independent of ε, we have that : $\mathcal{K}_\varepsilon \xrightarrow{\gamma} \mathcal{K}$. Then we can use Property 12 which gives us the expected result. □

This result is interesting because we do search for our model to be a function in $\mathcal{H}^{1,p}(\mathcal{X})$, this comes from Property 8. But the biggest issue here is that we need $p > n$ which in **high dimension** - this report is set in high dimension as the title mentions it - could mean that p is computationally infinite (image in high quality for example). And in our case this would mean that we are back to using Lipschitz basic regularization. Indeed if $p = +\infty$, then we are set in $\mathcal{H}^{1,\infty}(\mathcal{X})$ with the norm being the infinity one $\|f\|_\infty = \underset{x\in\mathcal{X}}{esse\ sup}\, f(x)$. This means that $\mathcal{E}(f) = \|\nabla f\|_{(L^p(\mathcal{X}))^n} = \|\nabla f\|_{\infty^n}$ which is basically the Lipschitz constant.

Hence, we would have preferred a result for $p = 2$. However $p = 2$ means that we cannot write for $x \in \mathcal{X}$, $f \in \mathcal{H}^{1,2}(\mathcal{X})$, $f(x)$ this does not make sense. Thus we could either revisit the part $\sum_{(x,y)\in\Gamma} loss(f(x), y)$ of the problem or change the space in which we have set our problem. Indeed we not only know that our models functions are in $\mathcal{H}^{1,p}(\mathcal{X})$ but we also know about their precise form which is neural networks form. Let see both solutions and the issues they could imply here.

Revisiting the *loss* :

If we do the following change :

$$\sum_{(x,y)\in\Gamma} loss(f(x), y) \approx \int_{\mathcal{X}} loss(f(x), y)\, d\Pi(x, y)$$

where we assume that Π is a continuous distribution for our data with respect to Lebesgue measure. Then we directly obtain, for all $p > 1$ we have in $\mathcal{H}^{1,p}(\mathcal{X})$:

$$\mathcal{K}_\varepsilon \xrightarrow{\gamma} \mathcal{K}$$

where $\mathcal{K}_\varepsilon$ and $\mathcal{K}$ are redefined as :

$$\mathcal{K}_\varepsilon(u) = \int_{\mathcal{X}} loss(f(x), y)\, d\Pi(x, y) + \lambda\mathcal{E}_\varepsilon(u)$$

$$\mathcal{K}(u) = \int_{\mathcal{X}} loss(f(x), y)\, d\Pi(x, y) + \lambda\mathcal{E}(u)$$

[31] It is important to understand that without this result writing $u(x)$ does not make sense.

But such a solution has issues.
Firstly, it is purely theoretical and thus can not be applied to our algorithm. Indeed, in applied case our data distribution would absolutely not be continuous but made of diracs on some data points.
Secondly, such a definition for our *loss*-term implies to fully know the data distribution which is not the case in most problems using neural networks.

Thus we are going to explore an other possibility.

Changing our space :

Using the Sobolev embedding theorem to obtain continuity is -in some way- exaggerated. Indeed, we already know that our neural networks are continuous due to their definition. So a more correct idea, would be to change the space in which we prove the minimizer convergence. There exist spaces made to represent neural networks functions. Those spaces are often called Barron spaces referring to Andrew R. Barron which was the first to define those spaces [Bar93]. The name was probably introduced in [Wei19] which gives an overview on those spaces.

Definition 4 (Barron Space of the First Type)**.** *A function f belongs to the Barron space of the first type if it can be written as:*

$$f(x) = \int_{\mathbb{R}^d \times \mathbb{R}} a(w,b)\sigma(w \cdot x + b)\, d\mu(w,b)$$

where μ is a probability measure on $\mathbb{R}^d \times \mathbb{R}$ and $a(w,b)$ is a measurable function such that the following norm is finite:

$$\|f\|_{\mathcal{B}} = \inf \left\{ \int_{\mathbb{R}^d \times \mathbb{R}} |a(w,b)|\, d\mu(w,b) : f(x) = \int_{\mathbb{R}^d \times \mathbb{R}} a(w,b)\sigma(w \cdot x + b)\, d\mu(w,b) \right\}$$

Definition 5 (Barron Space of the Second Type)**.** *Alternatively, a function f belongs to the Barron space of the second type if it satisfies:*

$$\|f\|_{\mathcal{B}'} = \inf \left\{ \mathbb{E}_{(w,b)\sim\mu}[|a(w,b)|] : f(x) = \mathbb{E}_{(w,b)\sim\mu}[a(w,b)\sigma(w \cdot x + b)] \right\} < \infty$$

where $\mathbb{E}$ represents the expectation with respect to the measure μ.

The main idea here would be to prove using those spaces that γ-convergence in $\mathcal{H}^{1,p}(\mathcal{X})$ implies weaker-convergence in Barron spaces. And then define a minimizer convergence (similar to γ-convergence) in those Barron spaces.

Weaker-convergence in Barron spaces is based on the Schwartz distribution theory[32] :

Definition 6 (Weak Convergence in Schwartz Distribution Theory)**.** *Let $\{T_n\}$ be a sequence of distributions in the sense of Schwartz distributions, and let T be another distribution. We say that $\{T_n\}$ converges weakly to T if for every test function $\varphi \in \mathcal{D}(\mathcal{X})$, the following holds:*

$$\lim_{n\to\infty} \langle T_n, \varphi \rangle = \langle T, \varphi \rangle$$

where $\langle T, \varphi \rangle$ denotes the action of the distribution T on the test function φ.

[32]This is possible because every Barron function is continuous which implies that it is $L^1_{loc}(\mathcal{X})$, thus admits a Schwartz distribution.

Such result will come later in the article that is being written on the works presented so far. We are still researching and the period of this internship does not permit to include those results in this report.

Thus, I will not explain further about Schwartz distribution theory. If you want to learn more on this, a complete and rigorous course introducing this theory is given in French at ENSTA Paris : MA102 [BBD21].

5 Results & Comparison

In this section, we will detail the results obtained using our different regularizers trained on the MNIST dataset.

5.1 Comparison under Adversarial Attacks

As described above, one way to estimate the robustness of a neural network model is by using adversarial attacks[33]. Here, we will compare our training methods under different adversarial attacks.
To do so, we trained 5 fully connected models for each training method and calculated the mean accuracy of these models on adversarially attacked data. This accuracy was computed for different attack intensities (Epsilon).
The results of this process are presented in Figure 8 and Figure 9.

The different training processes depicted in these figures are:

- **ADV**: Adversarial training, where the neural networks are trained using data modified by FGSM (Epsilon = 0.05).
- **SUM**: Training with the FLIP algorithm using the sum estimator, allowing us to evaluate how CLIP performs in this context.
- **TV**: Training with a random Total Variation algorithm (Algorithm 4).
- **pTV**: Training with a projected Total Variation algorithm (Algorithm 3).
- **STA**: Standard training without any regularizer.

First of all, it is important to note that the Epsilon axis is log-scaled.
FGSM: Figure 8 shows that Total Variation achieves significantly better results than CLIP. A noteworthy observation is that Total Variation, and FLIP methods in general, maintain good accuracy on data with low attack weights, similar to standard training. Indeed, the drawback of adversarial training is that increasing robustness often comes at the expense of general accuracy. Thus, for low attack intensities (Epsilon), adversarial training is easily outperformed by standard training. Total Variation improves accuracy at higher Epsilon values while maintaining impressive general accuracy. This indicates that such training is a good compromise, and with further development or more accurate algorithms for estimating total variation, we might expect even better results than with adversarial training.
PGD: Figure 9 presents a more nuanced picture. It shows that while TV offers an interesting balance between robustness and general accuracy, it is not always a clear-cut compromise. It may be possible to combine both methods to train a more robust model.

[33]For a detailed explanation of adversarial attacks, please refer to Section 7.3.

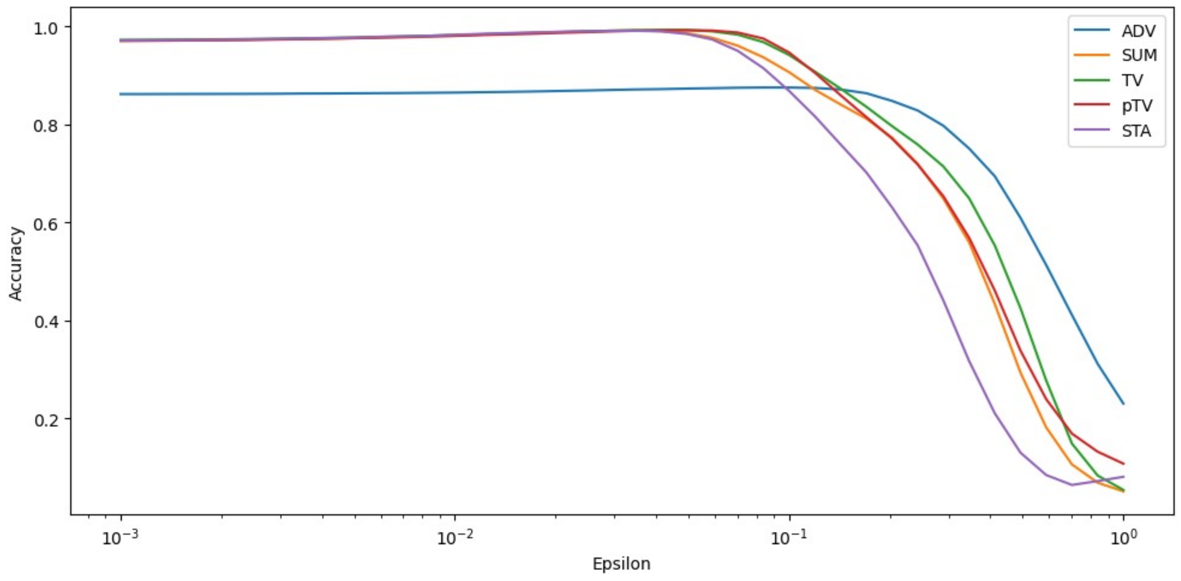


Figure 8: Comparison of accuracy against **FGSM** adversarial attacks for different training methods.

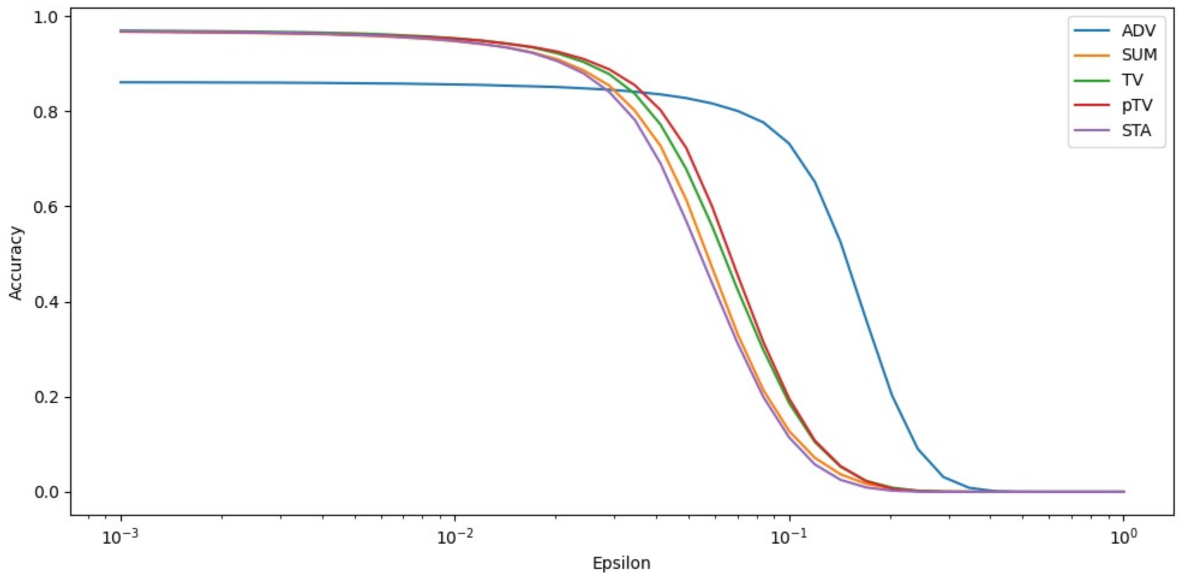


Figure 9: Comparison of accuracy against **PGD** adversarial attacks for different training methods.

5.2 Comparison under Noised Data

It is also valuable to examine how well a model performs when confronted with data that has been subjected to noise. In a way, this represents a type of attack that is not "adversarial" in nature. For this experiment, we added Gaussian noise to our images with varying intensities (Epsilon). This can be expressed by the following formula:

$$\hat{M_{ij}} = M_{ij} + \epsilon\,\mathcal{G}$$

where $\hat{M}$ is the matrix representing the noised image, M is the matrix representing the original image, $\mathcal{G}$ is the random variable representing the Gaussian noise, and ϵ is the scaling factor used to control the amount of noise added.

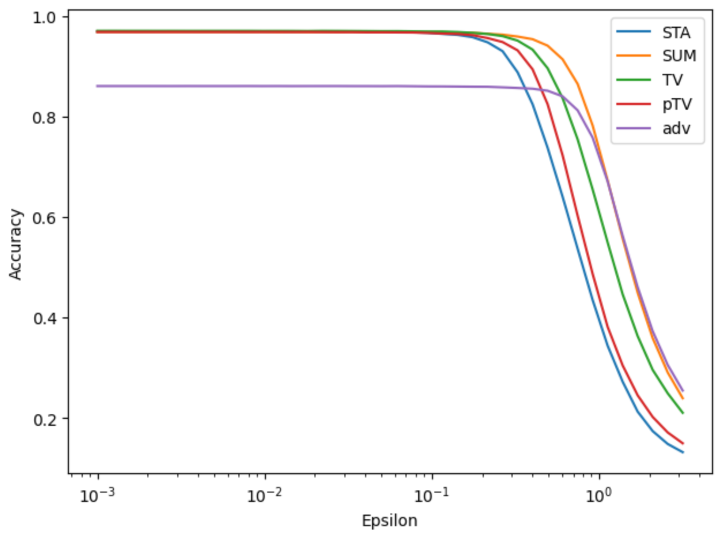


Figure 10: Comparison of accuracy against Gaussian noise for different training methods.

Using the same models trained in the previous section, we computed the results presented in Figure 10.

Figure 10 demonstrates that FLIP with the sum estimator achieves accuracy levels similar to adversarial training at high noise intensities. Remarkably, it does so while maintaining strong general accuracy. For lower noise intensities, the SUM method retains a high level of accuracy, unlike the adversarial approach.
This suggests that FLIP algorithms have a compelling effect: they smooth the model in a way that enhances robustness without sacrificing general accuracy.

5.3 Comparison of Training Time

During the training of the models used in the previous experiments, we recorded the computation time required for each method (using a CPU[34]: AMD EPYC 7402). This give us Table 3.

Estimator	Train Accuracy	Computation Time (s)
SUM FLIP	0.979	126
ADV FGSM	0.837	116
TV	0.981	95
pTV	0.978	94
STA	0.979	54

Table 3: Performance comparison of different training methods.

An interesting observation is that the TV method requires significantly less computation time than the SUM FLIP method, despite both relying on the FLIP (Finite Lipschitz) algorithm to estimate their primary regularization term. The key difference lies in the number of iterations needed to compute $\mathcal{X}lip$. For SUM FLIP, considerably more iterations are necessary because it involves computing over a broader set, while in the case of TV, this computation is restricted to a small ball around each point, which ultimately tends to a point as the process continues. This reduction in iterations leads to a notable decrease in computation time.

Furthermore, adversarial training (ADV FGSM) requires more time than both Total Variation (TV and pTV) methods,

[34]GPU architecture is often better to train NN but in our case, we were using the DESY server system.

although it is still faster than SUM FLIP. This underscores the efficiency of Total Variation methods, which not only deliver competitive accuracy but also offer a faster training process. The standard training method (STA) is the fastest, but this comes at the cost of lower robustness, as observed in earlier comparisons.

6 Conclusion

The exploration of robust training methods for neural networks through various regularization techniques has yielded compelling insights, particularly in the context of adversarial robustness and noise resilience. Throughout this work, we have systematically investigated the efficacy of different regularizers - MAX FLIP, SUM FLIP, Total Variation (TV), and their variants - culminating in a comparison with adversarial training methodologies.

The initial phase of our study involved validating the CLIP algorithm on a 1D problem, a crucial step that ensured the algorithm's stability and efficiency before scaling to higher dimensions. This step was not merely a formality but a necessary measure to uncover potential pitfalls in the algorithm's application, allowing for iterative improvements that would later manifest in higher-dimensional spaces.

In advancing the CLIP framework, we discovered and proposed a novel regularizer within the broader FLIP regularization family. The development of this new regularizer, which we carefully analyzed both empirically and theoretically, led to the creation of a neural network equivalent of Total Variation regularization. This development marks our contribution, as it extends the well-established concepts of TV regularization - widely recognized for its ability to preserve edges and reduce noise in image processing - into the realm of neural network training. The implications of this are profound, particularly when considering the role of Lipschitz continuity in enhancing the robustness of neural networks against adversarial perturbations and noisy data.

Our theoretical investigation into the proposed regularizer, underpinned by rigorous mathematical assumptions, provided a strong foundation for its practical application. By proving the existence of solutions under Total Variation regularization and demonstrating the behavior of models as the regularization weight tends to infinity, we have laid the groundwork for a deeper understanding of the interplay between regularization strength and model smoothness. The introduction of γ-convergence as a tool to analyze the limits of these algorithms further strengthens the theoretical underpinnings of our approach, offering a robust framework for future explorations in this domain.

The empirical results, particularly the comparison of model robustness under adversarial attacks and noisy data, have been revealing. The Total Variation-based methods often outperformed the SUM FLIP and adversarial training approaches in maintaining accuracy under both FGSM and PGD attacks, as well as in the presence of Gaussian noise. These findings highlight the potential of TV regularization as a powerful tool for enhancing the resilience of neural networks, particularly in scenarios where adversarial robustness is crucial but must be balanced with general accuracy.

This work opens several avenues for future research. Firstly, the development of more sophisticated algorithms to estimate Total Variation with higher accuracy could further enhance the performance of TV-based methods, potentially surpassing the robustness offered by adversarial training. Secondly, extending the theoretical analysis to encompass other forms of regularization within the FLIP family could provide new insights into the fundamental properties of these regularizers, leading to even more effective training methods. Lastly, applying these techniques to more complex, real world datasets - beyond the simplified settings of low quality images or 1D problems - would be a natural next step, testing the scalability and practical utility of our approaches in diverse and challenging domains.

7 Appendix

7.1 Notations & Properties

Notations :

Notation 1. $\mathcal{F}(\mathcal{X},\mathcal{Y})$ *: Functional space from* $\mathcal{X}$ *to* $\mathcal{Y}$*. We can write :* $\mathcal{F}(\mathcal{X},\mathcal{Y}) = \{f \mid f : \mathcal{X} \longrightarrow \mathcal{Y}\}$

Notation 2. $\subset$ *&* $\subsetneq$ *:*
$\subset$ *: We note* $A \subset B$ *if and only if* $\forall a \in A,\ a \in B$ *i.e.* $A \cup B = B$*.*
$\subsetneq$ *: We note* $A \subsetneq B$ *if and only if* $A \subset B$ *and* $\exists b \in B,\ b \notin A$ *i.e.* $A \cap B = A \neq B$*.*

Notation 3. $\mathcal{P}(\mathcal{X})$ *: Set of all* $\mathcal{X}$*-subsets. We can write :* $\mathcal{P}(\mathcal{X}) = \{\mathcal{U} \mid \mathcal{U} \subset \mathcal{X}\}$

Notation 4. ***A.E*** *: Stands for "for almost every". Thus, given property* P*, "**A.E*** $x \in \mathcal{X},\ P$*" means that there exists a measure* μ *such that there exists* $\tilde{\mathcal{X}}$ *such that* $\forall x \in \tilde{\mathcal{X}},\ P$ *and* $\mu(\mathcal{X}\backslash\tilde{\mathcal{X}}) = 0$*.*

Notation 5. $||\cdot||$ *: denotes an usual norm for the space our element is in. As we are always in finite dimension they are all equivalent (Property 17).*

Notation 6. $||\cdot||_2$ *: denotes a specific norm called euclidian norm. Computed as follows in a* R^n *space :* $\|\cdot\|_2 = \sqrt{\sum_{i=1}^n x_i^2}$*.*

Notation 7. $\Gamma_{\mathcal{X}},\ \Gamma_{\mathcal{Y}}$ *: denotes a projection of the* Γ *data space on* $\mathcal{X},\ \mathcal{Y}$*.*

Notation 8. $\nabla_x f(x')$ *: denotes the gradient of* f *over the coordinate* x *taken on the point* x'*.*

Notation 9. $\mathcal{J}_g(x)$ *: denotes the jacobian matrix of the function* g *taken on the point* x*.*

Notation 10. $\underset{x\in\mathbb{R}}{esse\ sup}$ *: This is the essential supremum. Which means the supremum for almost every point.*
i.e *:* $x_0 = \underset{x\in\mathbb{R}}{esse\ sup}$ *if and only if **A.E*** $x \in \mathbb{R},\ x_0 \leq x$*.*

Notation 11. $C^k(\mathcal{X})$ *: denotes the set of k-differentiable functions which are continuous on their* k^{th} *differential. We have that*

$$C^k(\mathcal{X}) = \{f : \mathcal{X} \mapsto \mathbb{R} \text{ such that } f \text{ is k-differentiable and } D^k_x f \text{ is continuous for each } x \in \mathcal{X}\}$$

We denote $C^0(\mathcal{X})$ *the continuous function set.*

Notation 12. $L^p(\mathcal{X})$*: denotes a Lebesgue space. We have that*

$$L^p(\mathcal{X}) = \left\{ f : \mathcal{X} \to \mathbb{R} \text{ measurable such that } \left(\int_{\mathcal{X}} |f|^p\, d\mathbb{L}\right)^{\frac{1}{p}} < +\infty \right\} / (\underset{\mathbb{L}}{=}),$$

where $\underset{\mathbb{L}}{=}$ *is the equivalence relation:* $f \underset{\mathbb{L}}{=} g$ *if and only if **A.E.*** $x \in \mathcal{X},\ f(x) = g(x)$*.*

Notation 13. $\partial^\alpha u$*: denotes a weaker partial derivative. It is an m-th order weaker partial derivative if:* $\forall \alpha \in \mathbb{N}^d$ *such that* $|\alpha| = \sum_{j=1}^d \alpha_j \leq m$*, we have* $\partial^\alpha u = \frac{\partial^{|\alpha|} u}{\partial x_1^{\alpha_1} \cdots \partial x_d^{\alpha_d}}$*, where weaker derivative stands for derivation in the sense of distributions.*

Notation 14. *$\mathcal{H}^{m,p}(\mathcal{X})$: denotes a Sobolev space with $(m,p) \in \mathbb{N} \times [1,+\infty]$ on $\mathcal{X}$. Such a space is defined for an open space $\mathcal{X} \subset \mathbb{R}^d$ as*

$$\mathcal{H}^{m,p}(\mathcal{X}) = \left\{ u \in L^p(\mathcal{X}) \;\middle|\; \forall \alpha \in \mathbb{N}^d,\ |\alpha| \le m,\ \partial^\alpha u \in L^p(\mathcal{X}) \right\}.$$

When $p = 2$, we often denote $\mathcal{H}^{m,2}(\mathcal{X}) = \mathcal{H}^m(\mathcal{X})$.[35]

Properties :

Property 15. *Given a function f_θ differentiable on $x' \in \mathcal{X}$ and a norm $||\cdot||$ differentiable everywhere else than 0 (for instance, the euclidian one). We have that $Sq_{lip}(f_\theta,\cdot,x') = (Lip(f_\theta,\cdot,x'))^2 = \left(\frac{||f_\theta(x')-f_\theta(\cdot)||}{||x'-\cdot||}\right)^2$ is a differentiable function on $\mathcal{X}\backslash\{x'\}$.*

Proof. Our ratio $x \mapsto \frac{1}{||x'-x||^2}$ is differentiable everywhere else than x'. The real problem here is for $||f_\theta(x) - f_\theta(x')||$ which is not differentiable on the set $Eq(f_\theta, x')$. That is why we added the square, thus $||f_\theta(x) - f_\theta(x')||^2$ is differentiable on $Eq(f_\theta, x')$. We compute the gradient :

$$\nabla_x \mathrm{Sq}_{lip}(f_\theta, x, x') = \frac{||x-x'||^2 \nabla_x(||f_\theta(x) - f_\theta(x')||^2) - ||f_\theta(x) - f_\theta(x')||^2 \nabla_x(||x-x'||^2)}{||x-x'||^4}$$

$$= \frac{||x-x'||^2 \mathcal{J}_{f_\theta}(x)^T (f_\theta(x) - f_\theta(x'))}{||x-x'||^2} - \frac{||f_\theta(x) - f_\theta(x')||^2 (x-x')}{||x-x'||^3}$$

Hence we have that for all $u \in Eq(f_\theta, x')$, $\nabla_x \mathrm{Sq}_{lip}(f_\theta, x, x') \underset{x\to u}{\longrightarrow} 0$, the gradient does exist for elements of $Eq(f_\theta, x')$. □

Property 16. ***(Lebesgue's dominated convergence theorem)*** *: Let $\{f_n\}$ be a sequence of measurable functions such that $f_n \to f$ almost everywhere on a measurable set A. Suppose there exists an integrable function g such that $|f_n(x)| \le g(x)$ **A.E** $x \in A$ and for all n. Then:*

1. *f is integrable, i.e., $f \in L^1(A)$.*

2. *The sequence of integrals of f_n converges to the integral of f:*

$$\lim_{n\to\infty} \int_A f_n \, dx = \int_A \lim_{n\to\infty} f_n \, dx = \int_A f \, dx.$$

Property 17. ***(Norm equivalence theorem)*** *: In a given normed vector space over a complete field we have that all norms are equivalent, i.e. given $\|\cdot\|_a$ and $\|\cdot\|_b$ there exist $M_a, M_b > 0$ such that $M_b\|\cdot\|_a \le \|\cdot\|_b \le M_a\|\cdot\|_a$.*

Property 18. *Let $f : \mathcal{X} \longrightarrow \mathcal{Y}$ be a lower semi-continuous and convex function. Suppose that $-\infty \in f(\mathcal{X})$. Then f is nowhere real-valued, i.e., $f(\mathcal{X}) \subset \{-\infty, +\infty\}$.*

Proof. This property is detailed and proved in this book : [Bau10] as Proposition 9.6. □

Property 19. ***(Bolzano-Weierstrass theorem)****: For a given normed $\mathbb{R}$-vectorial space and an infinite bonded sequence in such space, this sequence has a convergent subsequence.*

Property 20. *For $p > 1$ and for a function $f \in \mathcal{H}^{1,p}(\mathcal{X})$ on an open connected set $\mathcal{X}$ we have that if **A.E** $x \in \mathcal{X}$, $\nabla f(x) = 0$ then f is constant almost everywhere on $\mathcal{X}$.*

[35] I especially want to thank my professors Anne-Sophie Bonnet-Ben Dhia, Laurent Bourgeois, and Christophe Hazard for their very complete and rigorous course MA102 [BBD21] given at ENSTA Paris, which introduced the notations used here.

Proof. It was proved here math stack exchange that for an open connected subset $\mathcal{X} \subset \mathbb{R}^d$ a function $f \in \mathcal{H}^{1,p}(\mathcal{X})$ which satisfies $\partial^\alpha f = 0,\ \forall \alpha \in \mathbb{N}^d,\ |\alpha| \leq 1$ is constant almost everywhere on $\mathcal{X}$. An other way to prove this result could be to use Poincaré-Wirtinger inequality (or also Gagliardo–Nirenberg–Sobolev inequality). □

Property 21. ***(Kolmogorov-Hecht-Nielsen theorem [HN87])*** *A three-layer neural network can approximate any continuous multivariate function.*
i.e *: Let $\mathcal{X} \subset \mathbb{R}^n$ be a bounded set. For any continuous function $f : \mathcal{X} \to \mathbb{R}$ and any $\epsilon > 0$, there exists a neural network with one hidden layer and non-polynomial continuous activation function σ, such that the network function $\hat{f}$ satisfies:*

$$\sup_{x \in \mathcal{X}} |f(x) - \hat{f}(x)| < \epsilon.$$

Property 22. ***(Sobolev Embedding Theorem) :*** *Let $\mathcal{X} \subset \mathbb{R}^n$ be a bounded set. If $kp > n$, then any function $u \in \mathcal{H}^{k,p}(\mathcal{X})$ is also a continuous function on $\mathcal{X}$.*

Property 23. ***(Corollary) :*** *For the Sobolev space $\mathcal{H}^{1,p}(\mathcal{X})$ (first-order Sobolev space), to have $\mathcal{H}^{1,p}(\mathcal{X}) \subset C^0(\mathcal{X})$, we need $p > n$.*

7.2 Glossary

Gradient Often denoted by a nabla symbol (∇), representing the direction in which a function has the steepest ascent.

Lipschitz constant A constant that bounds how much a function's value can change with respect to changes in its input. The smaller this positive constant, the flatter the function.

Total Variation (TV) Represents the sum of all variations of a given function. It is linked to the gradient norm and the smoothness of a function.

Regularizer A term added to an optimization problem to enforce a constraint or to prevent overfitting.

MNIST A dataset of handwritten digit images.

Fashion-MNIST A dataset of Zalando's article images.

Adam An optimization algorithm that combines the advantages of two other extensions of stochastic gradient descent (SGD).

SGDM Stochastic Gradient Descent, an optimization method that updates parameters iteratively based on random subsets of data.

Lebesgue measure A method of assigning a volume to subsets of a n-dimensional space.

CLIP Cheap Lipschitz, a machine learning algorithm designed to handle high-dimensional data while regularizing the model using its Lipschitz constant.

Local & Global Maximum (Minimum) A point on which a function is maximal (minimal) locally or globally. A local maximum (minimum) is a maxima on a subset around our point.

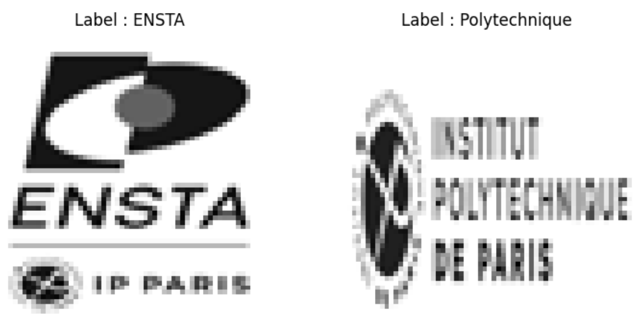


Figure 11: Image logo comparison with their labels (Sources: logos).

PGD (Projected Gradient Descent) [Mad18] We process using a gradient descent in the worst direction for our model while using projection to restrict the weight of such attacks.

FGSM (Fast Gradient Sign Method) [Goo15] ”It consists of adding a linear amount of in-perceivable noise to the image and causing a model to incorrectly classify it.” - Wikipedia.

Bounded A normed set is said to be bounded if for every element in the set, its norm is under a value $M > 0$ independent from this element.

7.3 Adversarial Training

Neural networks that have been trained on a "pure" dataset often lack robustness. Here, a "pure" dataset means untouched data.

To compute a model's robustness, we slightly perturb our data and test the model's accuracy on these perturbed data. It is important to understand that we compute a "robustness to" one kind of attacks. This does not give us an overall robustness which would be mathematically difficult to define.

Let's take an example:

We are going to conduct a small and amusing experiment (you can find the code detailed here: https://github.com/Raphael-Bernas/). We train a simple fully connected neural network on a small dataset made of logos. Some of the logos are labeled as "ENSTA" while others are labeled "Polytechnique". See Figure 11.

The idea is that our model returns a probability $f_\theta(x) = p \in [0, 1]$. If $p < 0.5$, we assume that the model has labeled the image x as "ENSTA"; otherwise, it is labeled as "Polytechnique".

Then, we train our model for 100 epochs using Cross-Entropy loss:

$$loss_{Cross}(\hat{y}, y) = -(\mathbf{1}_{\lfloor \hat{y} \rfloor = \lfloor y \rfloor} \log(\hat{y}) + (1 - \mathbf{1}_{\lfloor \hat{y} \rfloor = \lfloor y \rfloor}) \log(1 - \hat{y}))$$

$$\min_{\theta \in \Theta} \sum_{(x,y) \in \Gamma} loss_{Cross}(f_\theta(x), y)$$

After training, our model easily classifies each image correctly. Thus, we need to test its robustness. We are going to perturb an image so that the model misclassifies it. To do so, there exist many different "adversarial attacks." One of these is the Fast Gradient Sign Method (FGSM). Algorithm 5 details this method.

Algorithm 5: Fast Gradient Sign Method

```
Input  : $\epsilon$ the FGSM weight
Output: accuracy
for data $(x, y) \in \Gamma$ do
    $grad \leftarrow \nabla_x loss_{Cross}(f_\theta(x), y)$
    for $i = 1$ to $length(grad)$ do
        $GradSign_i \leftarrow \frac{grad_i}{|grad_i|}$
    $x_\epsilon^{adv} \leftarrow x + \epsilon\, GradSign$
    $accuracy \leftarrow accuracy + \mathbf{1}(x, y)_{\{\lfloor f_\theta(x) \rfloor = \lfloor y \rfloor\}}$
```

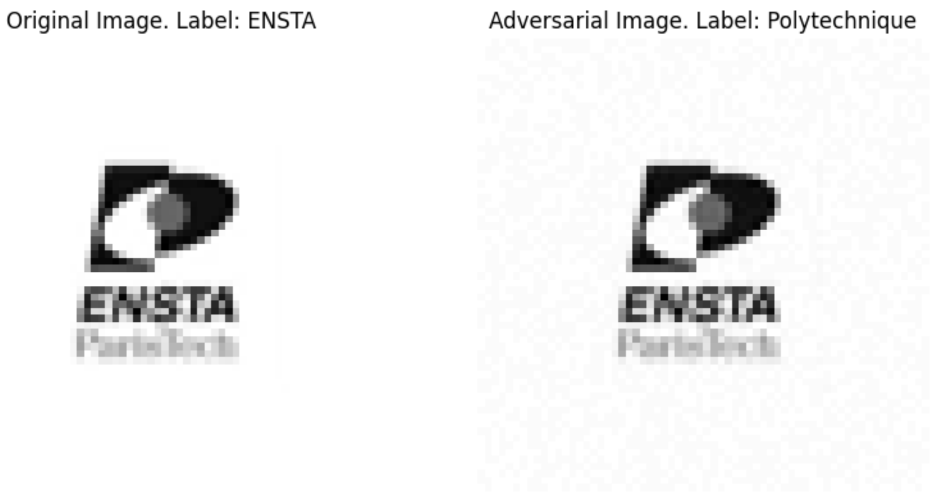

Original Image. Label: ENSTA
Adversarial Image. Label: Polytechnique
ENSTA
ParisTech
ENSTA
ParisTech

Figure 12: Almost imperceptible perturbed image is misclassified ($\epsilon \approx 2$ for my Git code).

The idea behind FGSM is to modify an image in the worst possible way for the model so that it misclassifies this image. Thus, we compute the gradient over x and not over θ of the *loss* function. This gives us the direction in which the *loss* function increases the most, which is also the direction in which the model performs the worst. Sometimes the perturbed image would even look exactly the same to human eyes but would still be misclassified. See in Figure 12 that the model indeed mislabeled our image. If you focus, you can see the small perturbation made by FGSM.

In fact, FGSM is like adding small noise to mislead the model. But at the same time, it is much more complex. Indeed, adding noise is most commonly done with random generation, which is not the case here. We choose to modify the data in the direction which is the worst for the model. Thus, even small noise can perturb the labeling.

This is why **adversarial training** has been developed—to increase robustness against such attacks. The main idea of adversarial training is to train the model on adversary-modified data. Algorithm 6 details such a training method.

Here are further insights on what happens for our model to make such mistakes:

- **Over-fitting**: Our model is overtrained on the data, making it perfect on those data but incapable on others. It is as if a student trained on a fixed list of questions and could not even answer those questions if you slightly changed them because they would not recognize that this is similar to the question they studied. Here, for example, changing the white background would completely confuse the model.

- **Overfocusing**: It is possible that there exists one pixel on which the model focuses most of its attention (it has a

Algorithm 6: Adversarial Training Method

```
for epoch $e = 1$ to $E$ do
    for minibatch $B \subset \Gamma$ do
        $B_{adv} \leftarrow \emptyset$
        for $x \in B$ do
            $x_{adv} \leftarrow AdversarialAttacks(x, \epsilon, f_\theta)$
                                                   // You could, for example, use FGSM
            $B_{adv} \leftarrow B_{adv} \cup \{x_{adv}\}$
        $\theta \leftarrow \mathrm{SGDM}_{\eta,\gamma}\left(\frac{1}{|B_{adv}|}\sum_{(x,y)\in B_{adv}} loss(f_\theta(x), y)\right)$
```

huge weight for the model). Then, just modifying this pixel could easily mislead the model.

7.4 Are our different regularizers truly different ?

This is a worry we could have, are we truly regularizing with different techniques ? Or isn't one of those techniques just obviously better ?

In this appendix subsection, we will seek to verify that none of the introduced regularizer is uninteresting in regard to one another. This obviously happens rarely, therefore this section does not truly have its place in the main part.

Firstly, let us define :

Definition 7. ***Regularizer:*** *A regularizer is a function reg* $: \mathcal{F}(\mathcal{X}, \mathcal{Y}) \times \mathcal{P}(\mathcal{X}) \longrightarrow \mathbb{R}^+$. *We will often write* $reg(f) := reg(f, \mathcal{X})$.

Then we define :

Definition 8. ***Equivalent Regularizers:*** *Given two regularizers* reg_1 *and* reg_2*, we call them equivalent on* $\mathcal{U} \subset \mathcal{X}$ *if and only if for all* $\lambda > 0$ *we have that*

$$\underset{\theta\in\Theta}{argmin}\{\sum_{(x,y)\in\Gamma} loss(f_\theta(x), y) + \lambda reg_1(f_\theta, \mathcal{U})\} = \underset{\theta\in\Theta}{argmin}\{\sum_{(x,y)\in\Gamma} loss(f_\theta(x), y) + \lambda reg_2(f_\theta, \mathcal{U})\}$$

We will denote such a relation as follows : $reg_1 \underset{solve\,\mathcal{U}}{\longleftrightarrow} reg_2$.

Sharper Regularizer: *Seemingly, we will call* reg_1 *sharper than* reg_2 *on* $\mathcal{U}$ *if and only if*

$$\underset{\theta\in\Theta}{argmin}\{\sum_{(x,y)\in\Gamma} loss(f_\theta(x), y) + \lambda reg_1(f_\theta, \mathcal{U})\} \subset \underset{\theta\in\Theta}{argmin}\{\sum_{(x,y)\in\Gamma} loss(f_\theta(x), y) + \lambda reg_2(f_\theta, \mathcal{U})\}$$

We will denote such a relation as $reg_1 \underset{solve\,\mathcal{U}}{\longrightarrow} reg_2$.

Remark 9. *We can define a relation called Strictly Sharper Regularizer relation as follows :* reg_1 *strictly sharper than* reg_2 *on* $\mathcal{U}$ *if and only if*

$$\underset{\theta\in\Theta}{argmin}\{\sum_{(x,y)\in\Gamma} loss(f_\theta(x), y) + \lambda reg_1(f_\theta, \mathcal{U})\} \subsetneq \underset{\theta\in\Theta}{argmin}\{\sum_{(x,y)\in\Gamma} loss(f_\theta(x), y) + \lambda reg_2(f_\theta, \mathcal{U})\}$$

Denoted $reg_1 \underset{solve\,\mathcal{U}}{\twoheadrightarrow} reg_2$

In the future, we will denote $reg_1 \underset{\text{solve } \mathcal{X}}{\longrightarrow} reg_2$ as $reg_1 \underset{\text{solve}}{\longrightarrow} reg_2$

Property 24. *The Equivalence Regularizer relation is an equivalence relation.*

Proof. To prove such result we need to prove that our relation is reflexive, symmetric and transitive. We will prove this over $\mathcal{X}$ but this does not impact the result which still holds for $\mathcal{U} \subset \mathcal{X}$.

Let us define three regularizers reg_1, reg_2 and reg_3 :

- Reflexivity : We obviously have that $reg_1 \underset{solve}{\longleftrightarrow} reg_1$.
- Symmetry : Suppose we have $reg_1 \underset{solve}{\longleftrightarrow} reg_2$.
  Then : $\underset{\theta\in\Theta}{argmin}\{\sum_{(x,y)\in\Gamma} \text{loss}(f_\theta(x),y) + \lambda reg_1(f_\theta,\mathcal{X})\} = \underset{\theta\in\Theta}{argmin}\{\sum_{(x,y)\in\Gamma} \text{loss}(f_\theta(x),y) + \lambda reg_2(f_\theta,\mathcal{X})\}$.
  Thus : $\underset{\theta\in\Theta}{argmin}\{\sum_{(x,y)\in\Gamma} \text{loss}(f_\theta(x),y) + \lambda reg_2(f_\theta,\mathcal{X})\} = \underset{\theta\in\Theta}{argmin}\{\sum_{(x,y)\in\Gamma} \text{loss}(f_\theta(x),y) + \lambda reg_1(f_\theta,\mathcal{X})\}$.
  Which confirms that $reg_2 \underset{solve}{\longleftrightarrow} reg_1$.
- Transitivity : Suppose $reg_1 \underset{solve}{\longleftrightarrow} reg_2$ and $reg_2 \underset{solve}{\longleftrightarrow} reg_3$. Then :

$$\underset{\theta\in\Theta}{argmin}\{\sum_{(x,y)\in\Gamma} \text{loss}(f_\theta(x),y) + \lambda reg_1(f_\theta,\mathcal{X})\} = \underset{\theta\in\Theta}{argmin}\{\sum_{(x,y)\in\Gamma} \text{loss}(f_\theta(x),y) + \lambda reg_2(f_\theta,\mathcal{X})\}$$

  Using $reg_2 \underset{solve}{\longleftrightarrow} reg_3$:

$$= \underset{\theta\in\Theta}{argmin}\{\sum_{(x,y)\in\Gamma} \text{loss}(f_\theta(x),y) + \lambda reg_3(f_\theta,\mathcal{X})\}$$

  Thus : $reg_1 \underset{solve}{\longleftrightarrow} reg_3$.

Hence our relation is an equivalence relation. □

Property 25. *The Sharper Regularizer relation is an order relation. With respect to antisymmetry defined as follows :* $reg_1 \underset{solve\,\mathcal{U}}{\longrightarrow} reg_2$ *and* $reg_2 \underset{solve\,\mathcal{U}}{\longrightarrow} reg_1$ *implies that* $reg_1 \underset{solve\,\mathcal{U}}{\longleftrightarrow} reg_2$.

Proof. To prove such result we need to prove that our relation is reflexive, antisymmetric and transitive.
This can be proved easily using that $\subset$ is an order relation for sets. □

Assumption 4. *Assume there exist $x \in \mathbb{R}^m$, $\nu > 0$ such that $\mathcal{B}_\nu(x) \subset \mathcal{X}$ and $\mathcal{B}_\nu(x) \cap \Gamma = \emptyset$.*

Assumption 5. *Assume that we are in the condition of Kolmogorov-Hecht-Nielsen Theorem 21, with our model f_θ having at least three layers.*

Property 26. *Neither $TV(f_\theta) = \int_{\mathcal{X}} \underset{\epsilon\to 0}{lim}\, Lip_\epsilon^{loc}(f_\theta,x)\,dx$ regularizer, nor $Lip(f_\theta) = \underset{x,x'\in\mathcal{X}}{sup} \frac{\|f_\theta(x)-f_\theta(x')\|}{\|x-x'\|}$ regularizer is sharper than one another. Thus, they are not equivalent.*

Proof. First we are going to prove that Lip is not sharper than TV :
Let $\lambda > 0$ suppose we have a θ^*_{lip} of $\underset{\theta\in\Theta}{min} \sum_{(x,y)\in\Gamma} \text{loss}(f_\theta(x),y) + \lambda Lip(f_\theta)$ assume that such a solution is also a solution to $\underset{\theta\in\Theta}{min} \sum_{(x,y)\in\Gamma} \text{loss}(f_\theta(x),y) + \lambda TV(f_\theta)$. We have that there exist $((x_n,x'_n))_{n\in\mathbb{N}} \in (\mathcal{X}^2)^{\mathbb{N}}$ which converge to (x_∞, x'_∞)[36] such that $\frac{\|f_\theta(x_n)-f_\theta(x'_n)\|}{\|x_n-x'_n\|} \underset{n\to+\infty}{\longrightarrow} Lip(f_\theta)$. There exists $x_0 \in \mathcal{X}$, such that $\exists\mu > 0$, $\exists N \in \mathbb{N}$, $\forall n > N$, $x_n, x'_n \notin \mathcal{B}_\mu(x_0)$. Indeed, for all $\epsilon > 0$ there exists a rank N_ϵ after which every $n > N_\epsilon$ we have that $\|x_\infty - x_n\| < \epsilon$ and $\|x'_\infty - x'_n\| < \epsilon$, then

[36]As $\mathcal{X}$ is bounded we can use Bolzano-Weierstrass, to facilitate the notation we will assume that our sequence is already the obtained subsequence.

using Assumption 4 we do have that if we choose $\epsilon < \frac{1}{4}\nu$, then we can find a point out of the ball[37] $\mathcal{B}_\epsilon(x_\infty)$ and the ball $\mathcal{B}_\epsilon(x'_\infty)$ for which there exists a ball around (thus we have such x_0 and μ existence) and with no data inside this ball. Let us define :

$$g : x \mapsto \begin{cases} \exp\left(-\frac{1}{1-\left(\frac{\|x-x_0\|}{r}\right)^2}\right) & \text{if } \|x - x_0\| < \mu, \\ 0 & \text{if } \|x - x_0\| \geq \mu. \end{cases}$$

We have that g is strictly positive on $\mathcal{B}_\mu(x_0)$ and null outside. Moreover, g is continuous (we can easily show that on the border of the ball, both inside and outside limits tend to 0) and is even Lipschitz continuous (ultimately it is even $\mathcal{C}^\infty(\mathcal{X})$). Indeed, the problem is on the border of the ball but we have that for all $x_1 \in \overline{\mathcal{B}_\mu(x_0)}/\mathcal{B}_\mu(x_0)$ and for all $p \in \mathbb{N}$:

$$\frac{1}{\|x - x_1\|^p} \exp\left(-\frac{1}{1 - \left(\frac{\|x-x_0\|}{r}\right)^2}\right) \xrightarrow[x \longrightarrow x_1]{} 0$$

Finally, let us define $L_g = Lip(g)$ and $\mathcal{F}_g = f_{\theta^*_{lip}} + \frac{Lip(f_{\theta^*_{lip}})}{4L_g} g$. Using Kolmogorov-Hecht-Nielsen Theorem 21 we obtain that there exists θ^*_{KHN}[38] such that $f_{\theta^*_{KHN}} = \mathcal{F}_g$ which is still an optimal solution for $\underset{\theta\in\Theta}{min} \sum_{(x,y)\in\Gamma} \text{loss}(f_\theta(x), y) + \lambda Lip(f_\theta)$[39] but is not one for $\underset{\theta\in\Theta}{min} \sum_{(x,y)\in\Gamma} \text{loss}(f_\theta(x), y) + \lambda TV(f_\theta)$ (because we strictly increased a local Lipschitz constant of our basic solution but not the global Lipschitz constant. Indeed, we are not on the point where the highest local Lipschitz constant was computed so the value did not change). This ultimately proves that if there exists a solution to Lip regularization that is also a solution to TV regularization, then there also exists a Lip solution which is not a TV solution. Thus proving the expected result.

Now we are going to give an idea on how to show that TV is not sharper than Lip :

Let suppose we still use the same λ and we have θ^*_{TV} a solution for TV regularization that is also one for Lip regularization. We denote $\partial\mathcal{X}$ the edge of $\mathcal{X}$ which is not empty because $\mathcal{X}$ is open and bounded (thanks to Assumption 3). On such a set we can define a function $g_{|\partial\mathcal{X}}$ for which the Lipschitz constant explodes. But as $\mathbb{L}(\partial\mathcal{X}) = 0$ we have obtained that $Lip(f_{\theta^*_{TV}} + g_{|\partial\mathcal{X}}) > Lip(f_{\theta^*_{TV}})$ and that $TV(f_{\theta^*_{TV}} + g_{|\partial\mathcal{X}}) = TV(f_{\theta^*_{TV}})$ which result in proving that if there exists a solution to TV regularization that is also a solution to Lip regularization, then there also exists a TV solution which is not a Lip solution[40]. Thus, we have proved the property. $\square$

Remark 10. *Property 26 gives a counter-example for Sharper relation being a total order relation.*

Remark 11. *What we proved with Property 26 gives us an idea of why* $SumLip(f_\theta) = \frac{1}{|\mathcal{X}_{lip}|} \sum_{(x,x')\in\mathcal{X}_{lip}} \frac{\|f_\theta(x)-f_\theta(x')\|}{\|x-x'\|}$ *and* $MaxLip(f_\theta) = \max_{(x,x')\in\mathcal{X}_{lip}} \frac{\|f_\theta(x)-f_\theta(x')\|}{\|x-x'\|}$ *are not equivalent (and neither one is sharper than the other).*

7.5 Vulgarization

Neural Networks consist of interconnected neurons that process inputs and generate outputs. These neurons are organized in layers, where each neuron in a layer receives inputs from the previous layer and passes its output to the next layer. This structured flow of information allows the network to learn complex patterns and make predictions (Find more on

[37] If we are in the case of being inside the ball defined in Assumption 4, if we are not then this ball itself can be chosen.

[38] This $f_{\theta^*_{KHN}}$ is - thanks to the theorem - a limit of functions $(f_{\theta_n})_n$ which were uniformly converging to $\mathcal{F}_g$. Thus, using the lower semi-continuity of all the problem terms we get that this is a minimizer for Lip regularizer problem.

[39] The *loss* function value did not change because we carefully made so that $g(\Gamma) = \{0\}$.

[40] Again, we can remark that the *loss* function value did not change thanks to the fact that obviously with Assumption 3 we have : $\Gamma \cap \partial\mathcal{X} = \emptyset$.

IBM Topics and Figure 13). To train neural networks, we require a substantial amount of data. First, the data we use

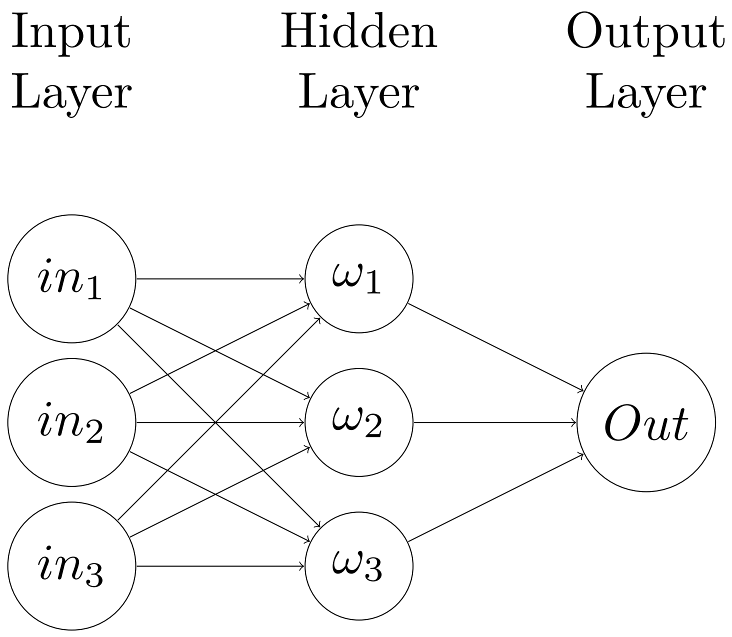


Figure 13: Simplified neural network (one hidden layer).

must contain both inputs and expected outputs. The training process proceeds as follows:

Training:

1. *Feed the data inputs to our model and obtain the outputs.*
2. *Compare the model's predicted outputs with the actual data outputs.*
3. *Adjust the neural network's weights ω based on the error. This step can be referred to as "punishing the model."*
4. *Repeat this training process on different batches of data.*
5. *Test the model on previously unseen data.*

However, the selection and training processes can lead to unsatisfactory or even dangerous outcomes. Below is a non-exhaustive list of common issues:

- **Over-fitting**: When the model is trained excessively, it fits the training data without generalizing well to new data. This results in excellent performance on the training set but poor performance on unseen data. (*Example: In a medical diagnosis application, an over-fitted neural network might predict a disease with high confidence based on specific features in the training set, even if those features are not generally indicative of the disease.*)
- **Bias**: When the training data contains biases, the model learns and perpetuates these biases. (*Example: In New York, a neural network used to predict crime hotspots was trained on biased historical crime data, resulting in higher predicted crime rates in over-policed neighborhoods, perpetuating a cycle of increased police presence and biased data collection* [Isa16].)
- **Concept Drift**: Models trained on data that changes over time can become outdated and perform poorly if not regularly retrained. (*Example: In spam detection, as the characteristics of spam emails evolve, a model trained on old data will degrade in performance unless updated regularly.*)

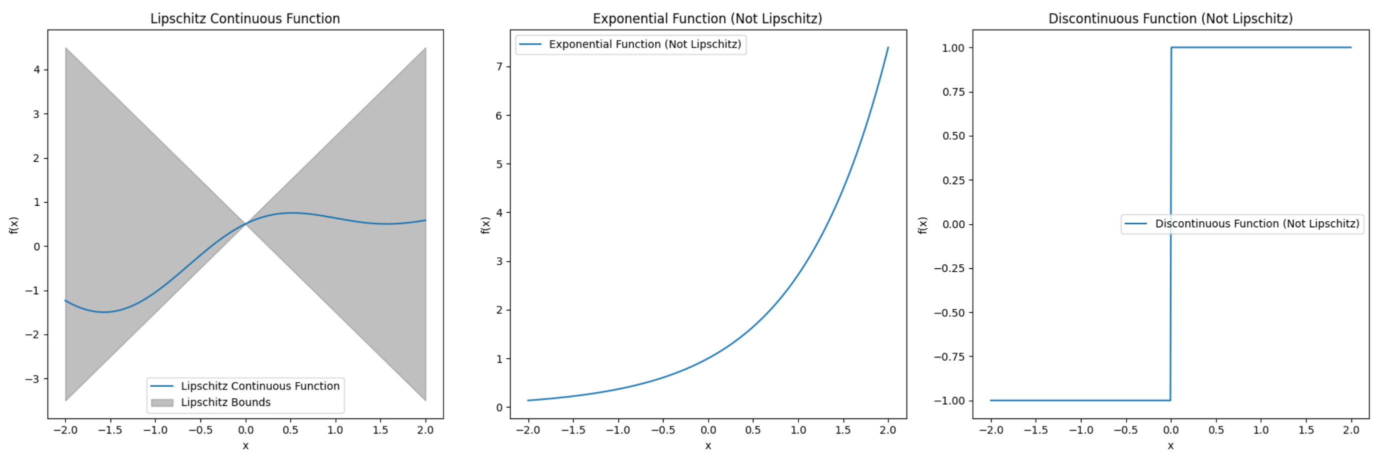


Figure 14: Example of lipschitz and non-lipschitz function

- **Adversarial Attacks**: Small perturbations to input data can drastically alter the model's output. (*Example: In image recognition systems, slight modifications to input images can fool a neural network into making incorrect predictions.*)

That is when regularization methods for neural networks become crucial. Indeed, regularization can be briefly described as an intelligent approach to processing data. Without regularization, the model focuses solely on improving accuracy on the training data, often at the expense of generalization.

The work presented above details the use of a regularization method called *Lipschitz regularization*.

The term *Lipschitz* is derived from the name of the renowned German mathematician *Rudolf Lipschitz*, after whom Lipschitz functions are named. A Lipschitz function is one where, for any given point, we can draw two lines passing through this point, and the rest of the function remains within the double cone defined by these lines, as illustrated in Figure 14. We call Lipschitz constant the highest value that the double cone inclination can take. But this constant can be infinite. Indeed, it happens when the double cone fills the whole space (The two lines are then vertical). However, a function with an infinite Lipschitz constant is not Lipschitz continuous, not a Lipschitz function (This is the case for the two other examples in Figure 14).

Lipschitz functions possess many useful properties that make them well-suited for manipulation. As illustrated by the examples in Figure 14, these functions do not exhibit explosive behaviors or discontinuities.

Regularizing our model with respect to its Lipschitz constant involves reducing this constant, thus preventing the model from exhibiting excessive variations or explosive growth. By enforcing a smaller Lipschitz constant, the model function becomes smoother and is able to generalize better on our data.

The result of this regularization is an increase in accuracy over test data (data other than those used for training) at the expense of a slight decrease in accuracy over the training data.

We can visualize regularizing the Lipschitz constant as reducing the cone in which the function resides. This visualization clearly highlights the primary issue of this regularization technique. If we regularize too much, the cone becomes narrower, and eventually, we might obtain a flat function, which is undesirable.

To prevent the regularization from flattening our function, we introduce a weight in our regularizer, denoted as λ. If λ is equal to zero ($\lambda = 0$), the regularizer is not applied. Conversely, if λ tends to infinity ($\lambda \to +\infty$), the resulting model becomes constant, meaning it would produce the same output for all inputs, which is clearly undesirable.

Figure 5 illustrates the need for careful selection of λ. If λ is too small, we end up with a high Lipschitz constant and over-fitting. On the other hand, if λ is too large, we lose precision on the data as our model becomes overly flattened. To implement this Lipschitz regularization, we use the method detailed in CLIP [Bun22], which stands for "Cheap Lipschitz". It is important to understand that the objective of such a method is to compute the Lipschitz constant. It has been proven in [Sca18] that computing this constant normally is NP-Hard. This essentially means that with our current computers, it would be unfeasible because the number of calculations required would explode quickly. The idea introduced in CLIP [Bun22] is to compute this Lipschitz constant more easily: we will search for the point at which the Lipschitz constant is the highest.

Indeed, the Lipschitz constant is, by definition, a supremum—which means the highest value or limit of values—over all values that the inclination of our double cone takes for every point. So finding a point at which the Lipschitz constant is the highest is not the perfect way, as a supremum could not be attained at a distinguished point but is a compromise between obtaining the correct value and computing it in a finite time (which is an understatement here, because for NP-Hard problems with our present computers, we would probably be dead before finding the exact Lipschitz constant).

An obvious question then would be: "Why would it be interesting to use such a method of searching for the highest point? And at what cost does this method give us a Lipschitz constant?" Indeed, why would it be easier to compute the highest point? The answer is that computing maximum and minimum is something a lot of past scientists have worked on, hence there exist very efficient algorithms and an important theory on those. One of the most basic algorithms used in this field is GDM, which stands for Gradient Descent Method. The idea behind this method is first computing a gradient, which is a vector, that we can represent as an arrow pointing in the direction in which the Lipschitz constant is the highest. Then, moving our point in this direction with a specified step length. By repeating this process, we hope to find the highest Lipschitz constant value computable at a point.

Figure 15 shows the evolution of such a process on a sample function. To be more precise, we need two points to compute an estimate of the Lipschitz constant (because we need two points to draw a line that defines our cone), hence we proceed the same way but over two points.

Now, to answer the second question—"At what cost?"—we need to detail one of the main issues of such methods: For most problems, including ours, this is a local method. This means that the point we find after many iterations gives us a local maximum of the Lipschitz constant but not necessarily the global maximum. It depends heavily on the starting point we use to compute the Lipschitz constant. In Figure 16, we illustrate this issue by comparing our algorithm to ants: our algorithm (ant) thinks the steepest direction leads to the highest hill but mistakenly ends up at a local maximum.

After computing an estimate of the Lipschitz constant, we penalize our model if this constant is too high. Through backpropagation, the model reduces its own Lipschitz constant. However, this reduction can cause the model to become flatter, potentially decreasing its accuracy in making correct predictions on the data. Consequently, in our algorithm, we adjust the weight λ that determines the extent to which the model is penalized for its Lipschitz constant.

If reducing the weight λ increases the accuracy on the data beyond a certain threshold, we then increase λ. To briefly explain the rationale behind this approach: as previously discussed, high accuracy on the training data might indicate that our model is over-fitting. By increasing the weight λ, we encourage the model to prioritize reducing its Lipschitz constant over maintaining high accuracy. Essentially, we penalize the model more for its Lipschitz constant than for its accuracy, relatively to the weight λ. Figure 1 shows that if we train a neural networks (green line) to approximate a polynomial function (Blue line) with lacking data (blue point), then the model's over-fitting makes it terrible at approximating our

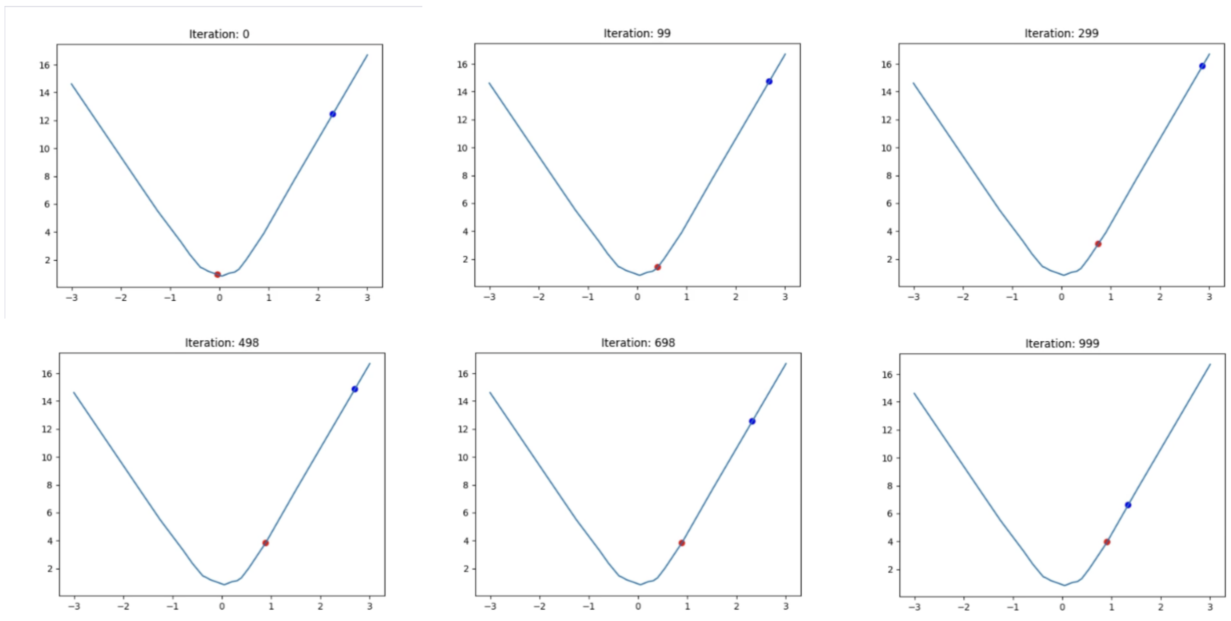


Figure 15: Evolution of a searched point on which the Lipschitz constant is the highest.

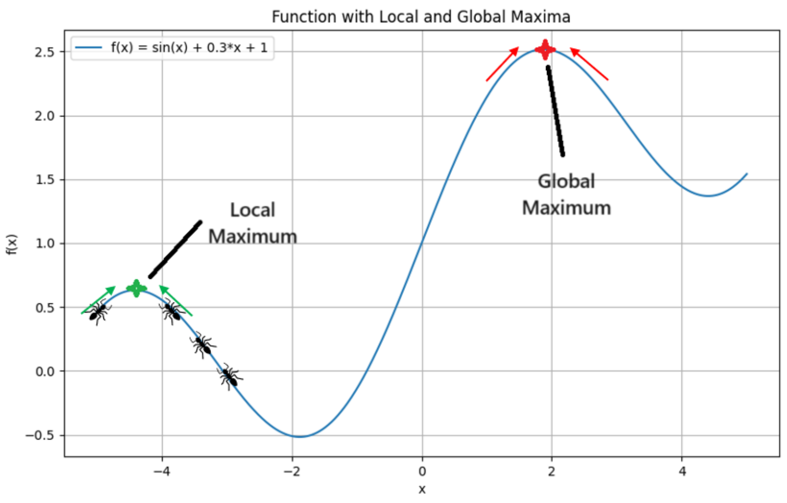


Figure 16: Example of a function with global and local maxima, where the ant mistakenly reaches the local maximum.

polynomial function. While using Lipschitz regularizer does make it better to approximate the function. We then confirm with a 1D experimentation (in a plan) that our Lipschitz regularization is indeed useful.

After confirming the efficiency of such a method, we ask ourselves if we could possibly make our algorithm faster, even if it is already cheap in term of computational time. The gradient descent method (GDM) we use is -as said before- one of the most basic optimizer methods to find an extremum. Which is why we are going to use Nesterov Accelerated Gradient (NAG) and ADAM optimizer. Both of them are extended versions of GDM made to be faster. In Figure 3 we compute for different learning rates (which is the step length we take in the direction given by the gradient) those three different optimizers. The result we obtain shows the three of them are not far in term of result while ADAM appears to be an interesting compromise between computation time and good accuracy.

Then, we make Remark 1 based on Figure 5 and result from [Bun22]. This remark detailed that we need to carefully choose the weight (λ) that we give to our Lipschitz regularizer in the training process. If too small, it doesn't impact much the training. But if too high, then our model becomes flatter.

High Dimension: Our primary objective in training neural networks is to process images. Images can be mathematically represented as matrices. A matrix is an array that contains many numbers, each representing a dimension. Therefore, we refer to this as high dimensionality. One of the most renowned image datasets, used for benchmarking models, is the *MNIST* dataset. This dataset consists of images of handwritten digits (serving as the inputs). Each image is labeled with a digit between 0 and 9 (serving as the outputs). The images in *MNIST* are matrices with 784 elements, corresponding to a dimension of 784, which is considered high dimensional. Figure 6 shows 9 images from the *MNIST* dataset that our model labeled correctly, while Figure 7 shows 9 images that were incorrectly labeled.
Using CLIP with different optimizers on the MNIST dataset provides the benchmark in Table 1. We observe that, to maximize accuracy, the Stochastic Gradient Descent (SGD) optimizer is preferable, whereas to minimize computation time, the ADAM optimizer should be used.
It is also beneficial to start our training without regularization until we reach a point of over-fitting. Then, we begin using our regularization. This approach can reduce the number of epochs needed.

Finite Lipschitz (FLIP): The algorithm used in CLIP allows us to compute an approximate maximum of the Lipschitz constant over a finite subset for different starting points (which, as explained in Figure 16, is important to consider due to possible local maxima). The proposed method involves taking the maximum among these computed local Lipschitz constant approximations. However, this approach loses all information about the local constants. Therefore, we introduce a new method that averages all computed Lipschitz constants. By doing so, we take into account different local Lipschitz constants.

As previously discussed, the Lipschitz constant is closely related to the variation of a function. Computing the sum of local Lipschitz constants is analogous to computing all variations. A well-known regularizer for image processing (not specifically for neural networks) is the ROF model [Rud92] (named after its authors Rudin, Osher, and Fatemi), which computes the sum of all variations. This regularizer is often referred to as Total Variation (TV) Regularization. It computes a supremum for a gradient, which is essentially like computing the limit of the local Lipschitz constant as the points get closer together. This is where the Finite Lipschitz (FLIP) algorithm becomes relevant. We aim to compute local Lipschitz constants for many points that become closer with each training step. Thus, we need an easy-to-compute algorithm, which is provided by the FLIP algorithm.

The main idea is to choose points that are at most a distance of ϵ apart, compute the highest sum of local Lipschitz constants around these points, and then gradually reduce ϵ so that it approaches 0. By doing this, we compute a limit. This method of computing Total Variation is more accurate for neural networks, as it regularizes over local Lipschitz constants.

Property for TV:

Firstly, we need to make some mathematical assumptions. Here I will describe them in terms of their utility.

- **Assumption 1**: Our *loss* function is proper. This means that the way we compare the model output and the expected output is correct.

- **Assumption 2**: Our neural network model must be Lipschitz continuous. This means that it needs to have a finite Lipschitz constant. We require small deformations of our model coefficients (weights) to have a small impact on the

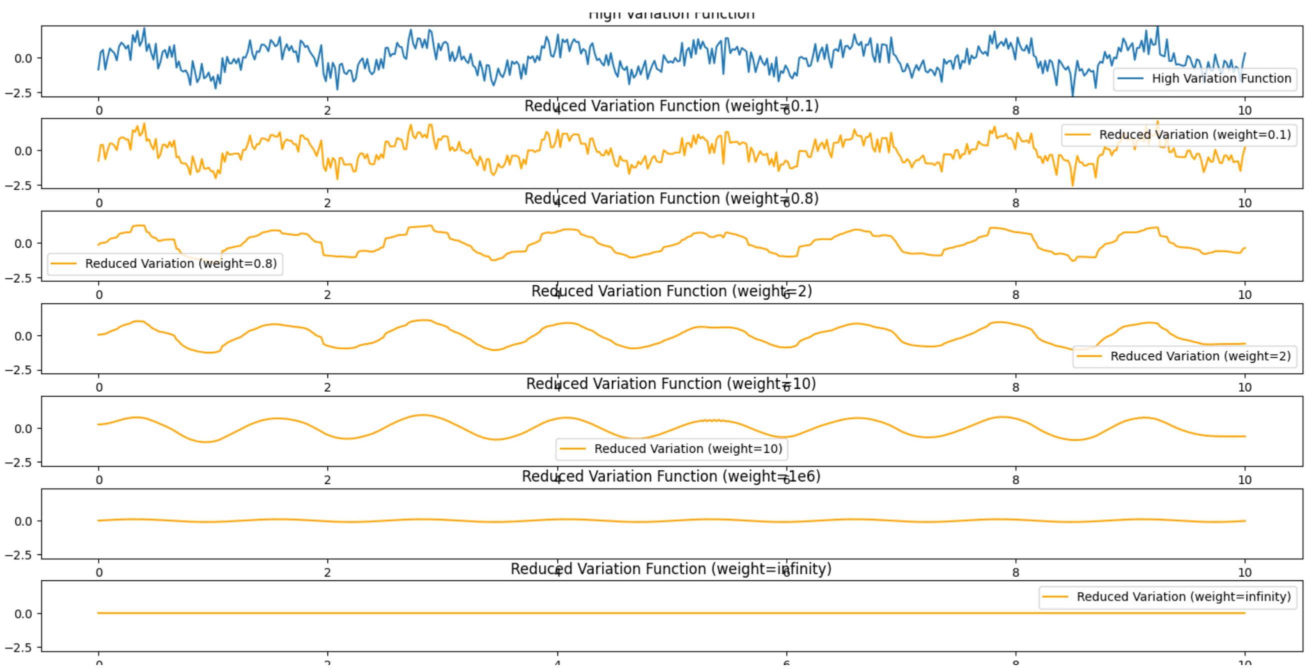


Figure 17: TV regularization with increasing weight λ to infinity.

model. Additionally, we need our model to have weights for which it is constant.

- **Assumption 3**: We need the set from which the data comes to be bounded, meaning we do not have exploding data values.

These assumptions are verified in the computer experiments we conduct, thus they can be assumed.

Under these assumptions, we can prove that ***there exists at least one solution for the TV problem***.

We can also prove that if we increase the weight of our TV regularization λ, then ***our solution model with infinite regularization is constant and tends to a barycenter of the data relative to the*** *loss* ***function definition***. Figure 17 illustrates this property.

The algorithm we implement for TV computes a solution for every distance ϵ with ϵ tending to 0. We expect our limits model—when ϵ tends to 0—to give us a solution for the TV basic problem computed over the gradient. However, this is not mathematically obvious, and we need to prove such a result. To do so, we have to use a weaker convergence, which is γ-convergence. This allows us to prove minimizer convergence without needing to add significant assumptions. This serves as a mathematical argument to use such an algorithm.

To conclude this vulgarization section I would suggest to read **Section 5:** Results & Comparison.

7.6 Context

7.6.1 Description

This section is meant to give further context on the internship that permitted this report.

ENSTA Paris: ENSTA Paris is a ”French Grande École” and part of the Institut Polytechnique de Paris. I am a student at ENSTA Paris majoring in Applied Mathematics. All students at ENSTA Paris undertake a research internship during their second year of study. After contacting Professor Burger, we agreed on my internship at DESY.

DESY (Deutsches Elektronen-Synchrotron): ”DESY is one of the world’s leading accelerator centers. Researchers use the large-scale facilities at DESY to explore the microcosm in all its variety – from the interactions of tiny elementary particles and the behavior of new types of nanomaterials to biomolecular processes that are essential to life. The accelerators and detectors that DESY develops and builds are unique research tools. The facilities generate the world's most intense X-ray light, accelerate particles to record energies, and open completely new windows onto the universe.

This makes DESY not only a magnet for more than 3000 guest researchers from over 40 countries every year, but also a coveted partner for national and international collaborations. Committed young researchers find an exciting interdisciplinary setting at DESY. The research center offers specialized training for a large number of professions. DESY cooperates with industry and business to promote new technologies that benefit society and encourage innovations. This also benefits the metropolitan regions of the two DESY locations, Hamburg and Zeuthen near Berlin" (Source: DESY Website).

DESY welcomed me to the *Computational Imaging Group* in Hamburg, led by Dr. Professor Martin Burger, who introduced me to the entire group. The *Computational Imaging Group* conducts research across the entire imaging process (measurements, preprocessing, and analysis) at DESY and other locations in Germany.

I've joined the Applied Mathematics Group which is part of the DESY and the Helmotz Imaging Group. This group mainly works on the preprocessing part. One of their research subject is to refine neural networks training for image processing. Thus, the subject of Lipschitz regularization we have worked on.

Internship : My internship permitted me to discover the whole DESY and its synchrotron, the cooperation between scientists from different fields, how we handle science research in maths and discovering all kinds of research done here at DESY. I have worked under Pr. Burger and Tim Roith guidance on the subject of Lipschitz regularization for neural networks. We worked to extend and dive further into what Tim and his other colleagues had already done in [Bun22]. During this internship, we had regular meetings where we could discuss the work my colleagues were doing. This truly gave me a broad perspective on the applied mathematics being done here.

I want to thank again Pr. Burger and Tim for providing me with this rich experience here in Hamburg. I won't ever forget the things I've seen, discovered and learned here.

7.6.2 Research process diagram

In Figure 18, every unit represents half a week. If a week begins with a day in month A but ends in month B then half the week is attached to month A and the other half attached to B.

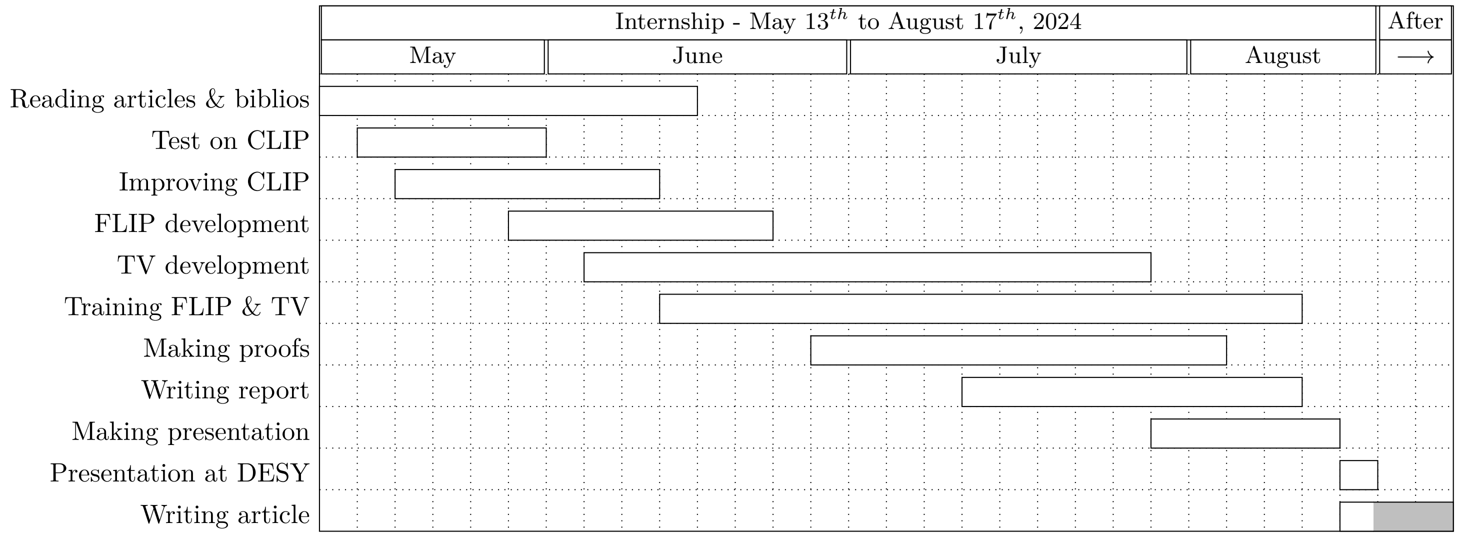


Figure 18: Time evolution of tasks during internship - Gantt diagram.

References

[Bar93] A. Barron. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information theory*, 39(3):930–945, (1993).

[Bau10] Combettes P. Bauschke, H. Convex analysis and monotone operator theory in hilbert spaces. *Springer*, (978-1-4419-9467-7), (2010).

[BBD21] Bourgeois L. Hazard C. Bonnet-Ben Dhia, A. Outils élémentaires d'analyse pour les equations aux dérivées partielles. *École d'ingénieur. Cours MIA 102 - Année 2022-2023, France.*, hal-04061707(pp.132.), (2021).

[Bos20] N. Bosse. An introduction to deep learning and the concept of regularization. *Published in 'Learning Deep' by Säfken, B., Silbersdorff, A., Weisser, C.*, (2020).

[Bun22] Raab R. Roith T. Schwinn L. Tenbrinck D. Bungert, L. Clip: Cheap lipschitz training of neural networks. *arXiv*, preprint arXiv:2103.12531, (2022).

[Cha09] Caselles V. Novaga M. Cremers D. Pock T. Chambolle, A. An introduction to total variation for image analysis. *HAL Science*, (hal-00437581), (2009).

[Cia09] Lunéville E. Ciarlet, P. La méthode des éléments finis: de la théorie à la pratique. tome 1: Concepts généraux. *hal-04039611*, 978-2-7225-0917-7.(pp.194), (2009).

[Get12] P. Getreuer. Rudin–osher–fatemi total variation denoising using split bregman. *in Image Processing On Line*, (2012).

[Goo15] Shlens J. Szegedy C. Goodfellow, I.J. Explaining and harnessing adversarial examples. *ICLR*, (2015).

[HN87] R. Hecht-Nielsen. Kolmogorov's mapping neural network existence theorem. *IEEE Int. Conf. on Neural Networks, IEEE Press, New York*, (1987).

[Isa16] Lum K. Isaac, W. To predict and serve? *The Royal Statistical Society*, (2016).

[Kin15] Lei Ba J. Kingma, D. P. Adam: A method for stochastic optimization. *ICLR*, (2015).

[Lat20] Rolland P. Cevher V. Latorre, F. Lipschitz constant estimation of neural networks via sparse polynomial optimization. *arXiv*, preprint arXiv:2004.08688, (2020).

[LeC98] Bottou L. Bengio Y. Haffner P. LeCun, Y. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278 - 2324, (1998).

[Mad18] Makelov A. Schmidt L. Tsipras D. Vladu A. Madry, A. Towards deep learning models resistant to adversarial attacks. *ICLR*, (2018).

[Rud92] Osher S. Fatemi E. Rudin, L. I. Nonlinear total variation based noise removal algorithms. *Physica*, D. 60 (1–4): 259–268., (1992).

[Sca18] Virmaux A. Scaman, K. Lipschitz regularity of deep neural networks: Analysis and efficient estimation. *In: NeurIPS*, (2018).

[Shi24] Burger M. Shi, K. Hypergraph p-laplacian regularization on point clouds for data interpolation. *arXiv*, preprint arXiv:2405.01109, (2024).

[Wei19] Chao M. Lei W. Weinan, E. The barron space and the flow-induced function spaces for neural network models. *arXiv*, preprint arXiv:1906.08039, (2019).

[Xia17] Rasul K. Volgraff R. Xiao, H. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv*, preprint arXiv:1708.07747, (2017).