On the Smoothing of Deep Networks

Vincent Roulet
Department of Statistics
University of Washington
Seattle, USA
vroulet@uw.edu

Zaid Harchaoui
Department of Statistics
University of Washington
Seattle, USA
zaid@uw.edu

***Abstract*—Many popular deep neural networks implement an input-output mapping that is non-smooth with respect to the network parameters. The non-smoothness has contributed to the difficulties in the theoretical analysis of deep learning. Several approaches have recently been proposed to address this specific difficulty at a great expense of mathematical sophistication. This work studies a simple approach consisting instead of smoothing the input-output mapping. We show how to perform the smoothing automatically within a differentiable programming framework. The impact of smoothing on the convergence behavior can be automatically controlled. We illustrate our approach with examples of multi-layer perceptrons and convolutional networks.**

***Index Terms*—deep neural network, first-order optimization, non-smooth optimization.**

I. Introduction

The input-output mapping implemented by a deep neural network is a chain of compositions of modules, where each module is typically a composition of a non-linear mapping, called an activation function, and an affine mapping. The last module in the chain is usually task-specific in that it relates to a performance accuracy for a specific task. This module can be expressed either explicitly in analytical form as in supervised classification or implicitly as a solution of an optimization problem as in dimension reduction or vector quantization.

Many popular deep neural networks use as an activation function a non-differentiable function such as the so-called rectified linear unit or ReLU function [8]. The resulting objective is then non-smooth with respect to the network parameters, and moreover, non-convex. These added difficulties have hindered the theoretical analysis of the convergence behavior of training algorithms from an information-based complexity viewpoint. Several finite-time guarantees were obtained under stringent assumptions and limited settings. Yet finite-time guarantees under general assumptions appear hard to obtain; the non-smoothness feature of the problems is an important roadblock [3, 4, 10].

We investigate here what happens if one considers instead smooth activation functions. First of all, there is a growing body of research suggesting that deep neural networks with smooth activation functions can perform just as well as their counterparts with ReLU activations in a number of tasks [6]. Second, even if one would like to train a deep neural network eventually with ReLU activation functions, one could still use smoothed counterparts for the purpose of training. Actually, a systematic approach can be built to perform smoothing of deep neural networks whose non-smoothness is due to the presence of non-smooth piecewise affine modules.

We show how to relate the training objective for a deep neural network with non-smooth modules to the one for its smoothed counterpart for any amount of smoothing. The impact of smoothing on the training objective can be computed automatically, given the smoothness constants of each module of the network. These automatic computations rely on the machinery of automatic differentiation, underlying the workings of training algorithms for deep neural networks [15, 13, 11]. See also [5, 2, 14, 8, 1, 12] for an exposition of gradient back-propagation algorithms and differentiable programming frameworks.

The approach allows us to obtain a convergence guarantee in terms of optimization accuracy for a deep neural network with non-smooth modules for a training algorithm that works with its smoothed counterpart only along the iterations. We illustrate the approach with examples of multi-layer perceptrons and convolutional neural networks. The theoretical analysis and the numerical results suggest that a smoothed counterpart of a deep neural network can be easily trained using smooth gradient-based optimization and that its sequence of iterates is close to the one of the original deep neural network, with non-smooth modules with a small amount of smoothing.

All the proofs and additional details can be found in the Appendix.

II. Problem setting

A feed-forward deep network of depth τ can be described as a transformation of an input x_0 into an output x_τ through the composition of τ blocks, called layers, illustrated in Fig. 1. Each layer is defined by a set of parameters. In general, (see Appendix Bfor a detailed decomposition), these parameters act on the input of the layer through an affine operation followed by a non-linear operation. Formally, the t^{th} layer can be described as a function of its parameters u_t and a given input x_{t-1} that outputs x_t as

$$x_t = \phi_t(x_{t-1}, u_t) = a_t(b_t(x_{t-1}, u_t)), \tag{1}$$

where, generally, b_t is linear in u_t, affine in x_{t-1} and a_t is non-linear.

This work was supported by NSF CCF-1740551, DMS- 2023166, and faculty research awards.

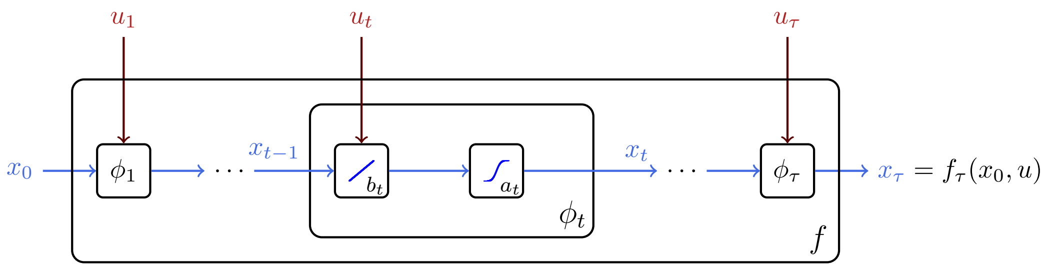

Fig. 1: Deep network structure.

Given a set of inputs outputs pairs $(x^{(i)}, y^{(i)})_{i=1}^n$, training a deep network consists in minimizing the distance between the given outputs $y^{(i)}$ and the outputs of the network applied to the inputs $x^{(i)}$ through a given loss ℓ. Formally, the problem is written

$$\begin{aligned} \min_{u_1,\ldots,u_\tau} \quad & \frac{1}{n}\sum_{i=1}^n \ell(y^{(i)}, x_\tau^{(i)}) + r(u_1,\ldots,u_\tau) \qquad (2)\\ \text{subject to} \quad & x_t^{(i)} = \phi_t(u_t, x_{t-1}^{(i)}) \quad \text{for } t = 1,\ldots,\tau \\ & x_0^{(i)} = x^{(i)}, \end{aligned}$$

where $u_t \in \mathbb{R}^{p_t}$ is the set of parameters at layer t whose dimension p_t can vary among layers and r is a regularization on the parameters of the network such as $r(u_1,\ldots,u_\tau) = \sum_{t=1}^\tau \lambda \|u_t\|_2^2/2$ with $\lambda \geq 0$.

We are interested in the influence of the structure of the problem, i.e., the chain of computations defined below and illustrated in Fig. 1, on the optimization complexity of the problem.

Definition 1. *A chain of τ computations $\phi_t : \mathbb{R}^{d_{t-1}} \times \mathbb{R}^{p_t} \to \mathbb{R}^{d_t}$ is defined as $f : \mathbb{R}^{d_0} \times \mathbb{R}^{\sum_{t=1}^\tau p_t} \to \mathbb{R}^{\sum_{t=1}^\tau d_t}$ such that for $x_0 \in \mathbb{R}^{d_0}$ and $u = (u_1;\ldots;u_\tau) \in \mathbb{R}^{\sum_{t=1}^\tau p_t}$ we have $f(x_0,u) = (f_1(x_0,u);\ldots;f_\tau(x_0,u))$ with*

$$f_t(x_0,u) = \phi_t(f_{t-1}(x_0,u), u_t) \quad \text{for } t = 1,\ldots,\tau, \qquad (3)$$

and $f_0(x_0,u) = x_0$. We denote $f_{t,x_0}(u) = f_t(x_0,u)$ and $f_{t,u}(x_0) = f_t(x_0,u)$.

Using this definition, problem (2) can be rewritten

$$\min_{u \in \mathbb{R}^{\sum_{t=1}^\tau p_t}} \frac{1}{n}\sum_{i=1}^n \ell(y^{(i)}, f_\tau(x^{(i)}, u)) + r(u), \qquad (4)$$

for f_τ defined by τ layers ϕ_t. Our goal is to understand how a deep network defined by non-smooth yet smoothable layers can be smoothed to any accuracy on a given bounded set. Smooth chains of computations are indeed amenable to theoretical studies of, e.g., the convergence rate of a gradient descent applied to (2).

We start by presenting how smoothness estimates of chains of computations can be computed automatically by having access to estimates of the smoothness properties of the layers. We then define what smoothable functions are and how a deep network can automatically be smoothed to any accuracy on any bounded set.

III. Smoothness of chains of computations

A. Notations

We present smoothness properties with respect to the Euclidean norm $\|\cdot\|_2$, whose operator norm is denoted $\|\cdot\|_{2,2}$. In the following, for a function $f : \mathbb{R}^d \to \mathbb{R}^n$ and a set $C \subset \operatorname{dom} f \subset \mathbb{R}^d$, we denote by

$$m_f^C = \sup_{x\in C} \|f(x)\|_2, \quad \ell_f^C = \sup_{\substack{x,y\in C\\ x\neq y}} \frac{\|f(x)-f(y)\|_2}{\|x-y\|_2},$$
$$L_f^C = \sup_{\substack{x,y\in C\\ x\neq y}} \frac{\|\nabla f(x)-\nabla f(y)\|_{2,2}}{\|x-y\|_2},$$

a bound of f on C, the Lipschitz-continuity parameter of f on C, and the smoothness parameter of f on C (i.e., the Lipschitz-continuity parameter of its gradient if it exists), all with respect to $\|\cdot\|_2$[1]. We denote by m_f, ℓ_f, L_f the same quantities defined on the domain of f, e.g., $m_f = m_f^{\operatorname{dom} f}$. We denote by $\mathcal{C}_{m,\ell,L}$ the class of functions f such that $m_f = m, \ell_f = \ell, L_f = L$. In the following, we allow the quantities m_f, ℓ_f, L_f to be infinite if for example the function is unbounded or the smoothness constant is not defined. The procedures presented below output infinite estimates if the combinations of the smoothness properties do not allow for finite estimates. On the other hand, they provide finite estimates automatically if they are available. In the following we denote $\bigotimes_{t=1}^\tau B_{R_t}(\mathbb{R}^{p_t}) = \{u = (u_1;\ldots;u_\tau) \in \mathbb{R}^{\sum_{t=1}^\tau p_t} : u_t \in \mathbb{R}^{p_t}, \|u_t\|_2 \leq R_t\}$.

B. Smoothness estimates

We present the smoothness computations for a deep network. Generic estimations of the smoothness properties of a chain of computation are presented in Appendix C. The estimation is done by a forward pass on the network, as illustrated in Fig. 2. The reasoning is based on the following lemma.

Lemma 2. *Consider a chain f of τ computations $\phi_t \in \mathcal{C}_{\ell_{\phi_t}, L_{\phi_t}}$ initialized at some $x_0 \in \mathbb{R}^{d_0}$.*

(i) We have $\ell_{f_{\tau,x_0}} \leq \ell_\tau$, where

$$\ell_0 = 0, \quad \ell_t = \ell_{\phi_t} + \ell_{t-1}\ell_{\phi_t}, \quad \text{for } t \in \{1,\ldots,\tau\}.$$

(ii) We have $L_{f_{\tau,x_0}} \leq L_\tau$, where

$$L_0 = 0, \quad L_t = L_{t-1}\ell_{\phi_t} + L_{\phi_t}(1+\ell_{t-1})^2, \quad \text{for } t \in \{1,\ldots,\tau\}.$$

[1]Note that if $x = \operatorname{Vec}(X)$ for a given matrix X, $\|x\|_2 = \|X\|_F$.

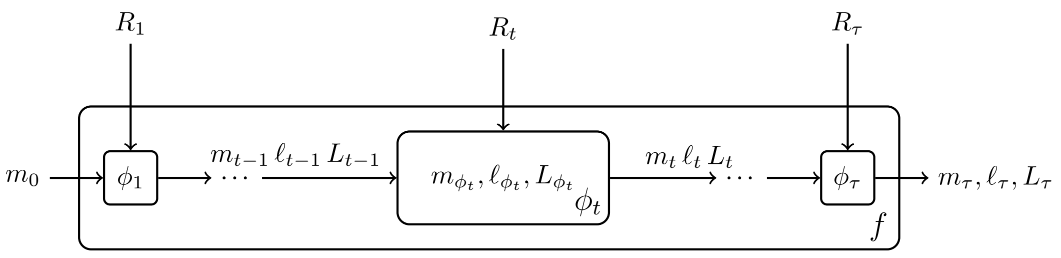
R_1
R_t
R_τ
m_0
ϕ_1
$m_{t-1}\,\ell_{t-1}\,L_{t-1}$
$m_{\phi_t}, \ell_{\phi_t}, L_{\phi_t}$
ϕ_t
$m_t\,\ell_t\,L_t$
ϕ_τ
$m_\tau, \ell_\tau, L_\tau$
f

Fig. 2: Smoothness estimates.

In the case of deep networks, the computations are not Lipschitz continuous due to the presence of bi-affine functions. Yet, provided that the inputs of the computations are bounded and that we have access to the smoothness of the computations, we can have an estimate of the Lipschitz-continuity of the computations restricted to these bounded sets. Formally, we have the following corollary.

Corollary 3. *Consider a chain f of τ of computations $\phi_t \in \mathcal{C}_{m_{\phi_t}, \ell_{\phi_t}, L_{\phi_t}}$ initialized at some $x_0 \in \mathbb{R}^{d_0}$ and consider $C = \bigotimes_{t=1}^{\tau} B_{R_t}(\mathbb{R}^{p_t})$. Then the smoothness of the output of the chain f_{τ,x_0} on C, can be estimated as in Lemma 2 by replacing ℓ_{ϕ_t} with $\tilde{\ell}_{\phi_t}$ defined by*

$$\begin{aligned}\tilde{\ell}_{\phi_t} &= \min\{\ell_{\phi_t}, L_{\phi_t}(m_{t-1}+R_t) + \|\nabla\phi_t(0,0)\|_{2,2}\},\\ m_t &= \min\{m_{\phi_t}, \tilde{\ell}_{\phi_t}(m_{t-1}+R_t) + \|\phi_t(0,0)\|_2\},\end{aligned}$$

for $t \in \{1,\ldots,\tau\}$, with $m_0 = \|x_0\|_2$.

We specialize the result for deep networks. We denote by $\mathcal{B}_{L,l^u,l^x}$ the set of L-smooth bi-affine functions b such that $\|\nabla_u b(0,0)\|_{2,2} = l^u$, $\|\nabla_x b(0,0)\|_{2,2} = l^x$, i.e., functions of the form

$$b(x,u) = \beta(x,u) + \beta^u(u) + \beta^x(x) + \beta^0,$$

with β bilinear and L-smooth, β^u, β^x linear and l^u, l^x Lipschitz continuous respectively and β^0 a constant vector.

Proposition 4. *Consider a chain f of τ computations whose layers ϕ_t are defined by*

$$\phi_t(x_{t-1}, u_t) = a_t(b_t(x_{t-1}, u_t)),$$

for $t \in \{1,\ldots,\tau\}$, where $b_t \in \mathcal{B}_{L_{b_t}, l^u_{b_t}, l^x_{b_t}}$, and a_t is decomposed as

$$a_t = a_{t,k_t} \circ \ldots \circ a_{t,1},$$

with $a_{t,i} \in \mathcal{C}_{m_{a_{t,i}}, \ell_{a_{t,i}}, L_{a_{t,i}}}$. Consider $C = \bigotimes_{t=1}^{\tau} B_{R_t}(\mathbb{R}^{p_t})$. The outputs m_τ, ℓ_τ and L_τ of Algo. 1 satisfy $m^C_{f_{\tau,x_0}} \le m_\tau$, $\ell^C_{f_{\tau,x_0}} \le \ell_\tau$, $L^C_{f_{\tau,x_0}} \le L_\tau$.

The proof of the above lemma is the consequence of simple technical lemmas provided in Appendix C. Note that the smoothness of the chain with respect to its input given a fixed set of parameters can also easily be estimated by a similar method; see Corollary 20 in Appendix C.

Algorithm 1 Automatic smoothness computations

```
Inputs:
1) Chain of computations $f$ defined by $\phi_t = a_t \circ b_t$ for $t \in \{1,\ldots,\tau\}$ with $a_t = a_{t,k_t} \circ \ldots \circ a_{t,1}$
2) Smoothness properties $L_{b_t}, l^u_{b_t}, l^x_{b_t}$ of the biaffine function $b_t \in \mathcal{B}_{L_{b_t}, l^u_{b_t}, l^x_{b_t}}$
3) Smoothness properties $m_{a_{t,i}}, \ell_{a_{t,i}}, L_{a_{t,i}}$ of the nonlinear functions $a_{t,i} \in \mathcal{C}_{m_{a_{t,i}}, \ell_{a_{t,i}}, L_{a_{t,i}}}$
4) Initial point $x_0$
5) Bounds $R_t$ on the parameters
Initialize $m_0 = \|x_0\|_2$, $\ell_0 = 0$, $L_0 = 0$
for $t = 1,\ldots,\tau$ do
    $\ell^x_{t,0} = L_{b_t}R_t + l^x_{b_t}, \quad \ell^u_{t,0} = L_{b_t}m_{t-1} + l^u_{b_t}, \quad \ell^0_{t,0} = 1$
    $m_{t,0} = \ell^x_{t,0}m_{t-1} + \ell^u_{t,0}R_t + \|b_t(0,0)\|_2$
    $L_{t,0} = 0$
    for $j = 1,\ldots,k_t$ do
        $\tilde{\ell}_{a_{t,j}} = \min\{\ell_{a_{t,j}}, \ \|\nabla a_{t,j}(0)\|_2 + L_{a_{t,j}}m_{t,j-1}\}$
        $m_{t,j} = \min\{m_{a_{t,j}}, \ \|a_{t,j}(0)\|_2 + \tilde{\ell}_{a_{t,j}}m_{t,j-1}\}$
        $\ell^0_{t,j} = \tilde{\ell}_{a_{t,j}}\ell^0_{t,j-1}$
        $L_{t,j} = L_{t,j-1}\ell_{a_{t,j}} + L_{a_{t,j}}(\ell^0_{t,j-1})^2$
    end for
    $m_t = m_{t,k_t}$
    $\ell_t = \ell^x_{t,0}\ell^0_{t,k_t}\ell_{t-1} + \ell^u_{t,0}\ell^0_{t,k_t}$
    $L_t = L_{t-1}\ell^x_{t,0}\ell^0_{t,k_t} + (L_{b_t}R_t + l_{b_t})^2 L_{t,k_t}\ell^2_{t-1}$
        $+2\left((L_{b_t}m_{t-1}+l^u_{b_t})(L_{b_t}R_t+l^x_{b_t})L_{t,k_t} + L_{b_t}\ell^0_{t,k_t}\right)\ell_{t-1}$
        $+(L_{b_t}m_{t-1}+l^u_{b_t})^2 L_{t,k_t}$
end for
Output: $m_\tau$, $\ell_\tau$, $L_\tau$
```

IV. Smoothing chains of computations

A. Smoothable functions

We define below smoothable functions.

Definition 5. *We say that a function $f \in \mathcal{C}^C_\ell$ is (L,K)-smoothable on a set C if, for any $\mu > 0$, there exists an approximation $f_\mu \in \mathcal{C}^C_\ell$ of f on C such that*

1) *for any $x \in C$, we have $\|f_\mu(x) - f(x)\|_2 \le \mu$,*
2) *for any $x \in C$, f_μ is differentiable with $L + K/\mu$-Lipschitz-continuous gradients.*

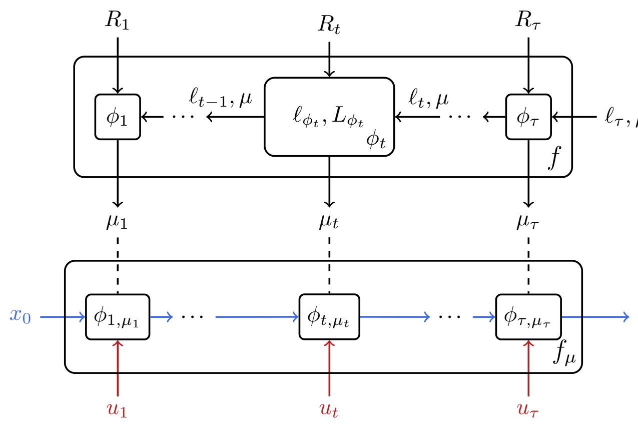

Fig. 3: Automatic smoothing for a target accuracy μ.

We denote by $\mathcal{S}^C_{\ell,L,K}$ the subset of functions in $\mathcal{C}^C_\ell$ that are (L, K)-smoothable on C. We denote by $\mathcal{S}_{\ell,L,K}$ the subset of functions in $\mathcal{C}_\ell$ that are (L, K)-smoothable on their domain of definition.

Note that any piece-wise affine function is smoothable. Moreover, note that $\mathcal{C}_{\ell,L} \subset \mathcal{S}_{\ell,L,0}$. By convention for $f \in \mathcal{C}_{\ell,L} \subset \mathcal{S}_{\ell,L,0}$, we denote $f_\mu = f$ for $\mu = 0$.

Example 6. *The ReLU function $\alpha(x) = \max(0, x)$ is $(0, 1/2)$-smoothable by defining for $\mu > 0$,*

$$\alpha_\mu(x) = \max_{z\in[0,1]} zx - \mu z^2 = \begin{cases} 0 & \text{if } x \leq 0 \\ \frac{x^2}{4\mu} & \text{if } 0 \leq x \leq 2\mu \\ x - \mu & \text{if } x \geq 2\mu \end{cases},$$

which is $1/(2\mu)$-smooth as the conjugate of a 2μ-strongly convex function and we have that $a_\mu(x) \leq a(x) \leq a_\mu(x) + \mu$ for all $x \in \mathbb{R}$.

In Appendix C, the smoothing of sums, concatenations, or compositions of smoothable functions is presented.

B. Smoothable chains of computations

Given a deep network whose non-linear functions are Lipschitz-continuous, the smoothing of the deep network on bounded sets is given by the following lemma and illustrated in Fig. 3.

Proposition 7. *Consider a chain f of τ computations ϕ_t defined by*

$$\phi_t(x_{t-1}, u_t) = a_t(b_t(x_{t-1}, u_t)),$$

for $t \in \{1, \ldots, \tau\}$, where $b_t \in \mathcal{B}_{L_{b_t}, l^u_{b_t}, l^x_{b_t}}$ and a_t is decomposed as

$$a_t = a_{t,k_t} \circ \ldots \circ a_{t,1},$$

with $a_{t,i} \in \mathcal{S}_{\ell_{a_{t,i}}, L_{a_{t,i}}, K_{a_{t,i}}}$. Consider $C = \bigotimes_{t=1}^\tau B_{R_t}(\mathbb{R}^{p_t})$. For any accuracy $\mu > 0$, Algo. 2 defines a smooth chain of computations f_μ, whose output approximates the output of f up to μ accuracy on C, i.e.

$$\forall x_0, u \in \mathbb{R}^{d_0} \times C, \qquad \|f_{\mu,\tau}(x_0, u) - f_\tau(x_0, u)\|_2 \leq \mu.$$

The smoothness parameter of f_μ is of the form $L + K/\mu$, where L, K can be computed by Algo. 1 (see proof for an analytical estimate).

Algorithm 2 Automatic smoothing

```
Inputs:
1) Chain of computations $f$ defined by $\phi_t = a_t \circ b_t$ for
   $t \in \{1, \ldots, \tau\}$ with $a_t = a_{t,k_t} \circ \ldots \circ a_{t,1}$
2) Smoothness properties $L_{b_t}, l^u_{b_t}, l^x_{b_t}$ of the bi-affine func-
   tion $b_t \in \mathcal{B}_{L_{b_t}, l^u_{b_t}, l^x_{b_t}}$
3) Smoothness properties $m_{a_{t,i}}, \ell_{a_{t,i}}, L_{a_{t,i}}, K_{a_{t,i}}$ of the
   smoothable functions $a_{t,i} \in \mathcal{S}_{m_{a_{t,i}}, \ell_{a_{t,i}}, L_{a_{t,i}}, K_{a_{t,i}}}$
4) Bounds $R_t$ on the parameters
5) Precision accuracy $\mu > 0$
Initialize $\ell_\tau = 1$, $n = 0$
for $t = \tau, \ldots, 1$ do
    $\ell_{t,k_t+1} = \ell_t$
    for $j = k_t, \ldots, 1$ do
        if $K_{a_{t,j}} > 0$ then
            $n := n + 1$
            $\mu_{t,j} = \mu/\ell_{t,j+1}$
        else
            $\mu_{t,j} = 0$
        end if
        $\ell_{t,j} = \ell_{a_{t,j}} \ell_{t,j+1}$
    end for
    $\ell_t = \ell_{t,1}(R_t L_{b_t} + l^x_{b_t})$
end for
Output: Chain of computations $f_\mu$ defined by $\phi_t = \tilde{a}_t \circ b_t$
for $t \in \{1, \ldots, \tau\}$ with $\tilde{a}_t = a_{t,k_t,\mu_{t,k_t}/n} \circ \ldots \circ a_{t,1,\mu_{t,1}/n}$
```

C. Consequences in terms of optimization

Consider now problems of the form

$$\min_{u \in \mathbb{R}^{\sum_{t=1}^\tau p_t}} F(u) := \frac{1}{n} \sum_{i=1}^n h^{(i)}(f_\tau(x^{(i)}, u)), \tag{5}$$

as in (4) (with $h^{(i)}(\hat{y}) = \ell(y^{(i)}, \hat{y})$ and ignoring the regularization for now).

If the chain f is composed of non-smooth Lipschitz-continuous functions, convergence guarantees for this problem are, up to our knowledge, only asymptotic [3]. If the computations ϕ_t of the chain f are smoothable, then by Prop. 7 the objective in (5) can be smoothed up to any accuracy on any bounded sets. Formally, we have the following lemma.

Lemma 8. *Consider the objective (5), assume the chain of computations f satisfies the assumptions of Prop. 7, and that $h^{(i)}$ is ℓ_h-Lipschitz continuous and L_h-smooth for any i. Consider $C = \bigotimes_{t=1}^\tau B_{R_t}(\mathbb{R}^{p_t})$. Then for any accuracy $\mu > 0$, the objective*

$$F_\mu(u) = \frac{1}{n} \sum_{i=1}^n h^{(i)}(f_{\nu,\tau}(x^{(i)}, u)),$$

Layer	1	2	3	4
Param. norm	36.84±0.09	19.01±0.19	36.76±0.07	5.24±0.74
Smoothing	$5 \cdot 10^{-6}$	$2 \cdot 10^{-4}$	$2 \cdot 10^{-2}$	0

TABLE I: Mean and standard deviation of the norm of the parameters of the MLP for the 50 first iterations of a non-smooth network. Smoothing used for a target accuracy $\mu = 1$.

Layer	1	2	3	4
Param. norm	4.17±1.21	3.90±1.23	4.46±1.17	4.44±1.21
Smoothing	$2 \cdot 10^{-6}$	$2 \cdot 10^{-4}$	$2 \cdot 10^{-2}$	0

TABLE II: Mean and standard deviation of the norm of the parameters of the ConvNet for the 50 first iterations of a non-smooth network. Smoothing used for a target accuracy $\mu = 1$.

with $\nu = \mu/\ell_h$, and $f_{\nu,\tau}$ computed by Algo. 2, is smooth on C and approximates F to μ accuracy on C, i.e., $|F(u) - F_\mu(u)| \leq \mu$ for any $u \in C$.

Convergence rates of first-order methods on smooth functions can then be applied for F_μ defined in Lemma 8. A projected gradient descent applied on F_μ constrained to C converge to an ε-stationary point in at most

$$\mathcal{O}\left(\frac{L^C_{F_\mu}(F_\mu(u_0) - F^*_\mu)}{\varepsilon^2}\right),$$

where $L^C_{F_\mu}$ is the smoothness constant of F_μ that can be approximated as $L^C_{F_\mu} \leq L^C_{f_\nu}\ell_h + L_h(\ell^C_{f_\nu})^2$ [7]. Similarly, a projected stochastic gradient descent whose variance of gradient estimators is bounded by σ^2, converges to an $\varepsilon + \sigma$-stationary point in the same maximal number of iterations [7].

If these first-order methods converge to a local minimum when applied to the smoothed objective F_μ then the overall procedure (smoothing + first-order method) gives access to a point that is also close to being a local minimum. Formally we have the following straightforward lemma.

Lemma 9. *Consider the assumptions of Lemma 8 and let $\varepsilon > 0$. Assume that a first-order method applied to F_ε on C outputs a point $\hat{u}$ that is ε minimal on its neighborhood. Namely, there exists $\eta > 0$ such that*

$$F_\varepsilon(\hat{u}) - \min_{u \in B_\rho} F_\varepsilon(u) \leq \varepsilon,$$

with $B_\rho = \{u \in \mathbb{R}^{\sum_{t=1}^\tau p_t} : \|\hat{u} - u\|_2 \leq \eta\} \subset C$. Then the point $\hat{u}$ is also 3ε minimal for the original objective F on this neighborhood, i.e., we have

$$F(\hat{u}) - \min_{u \in B_\rho} F(u) \leq 3\varepsilon.$$

V. Numerical Experiments

The purpose of the examples below is to illustrate the difference in terms of optimization trajectory between training a non-smooth deep-network and its smoothed counterpart. Namely, we are interested in

1) The gain/loss in terms of accuracy of the smoothed networks compared to the original non-smooth networks.
2) The difference in terms of the optimization path of the first-order methods applied to the non-smooth or smooth network.

A. Experimental setting

In both settings, we consider the full dataset, a squared norm regularization with coefficient $\lambda = 10^{-6}$, and a training objective with a logistic loss.

a) Multi-Layer Perceptron (MLP) on MNIST: We consider an MLP and the MNIST dataset. The MLP is defined by fully connected layers followed by the ReLU activation function except for the last one. We consider 3 hidden layers of size (4000, 1000, 4000).

b) Convolutional Network (ConvNet) on CIFAR10: We consider a ConvNet summarized as

Conv[16×32×32]→ReLU→Pool[16×16×16]
→Conv[20×16×16]→ReLU→ Pool[20×8×8]
→Conv[20×8×8]→ReLU→Pool[20×4×4]→FC

where Conv$[K, D, D]$ is a convolutional layer (with bias) with K filters that output an image of size $D \times D$, Pool$[K, D, D]$ is an average pooling operation with K, D, D defined as for the convolutional layer (stride and sizes of the patches are chosen such that the size of the image is reduced), and FC is a fully connected layer.

c) Optimization method: We consider a stochastic gradient method with mini-batch sampling with size 256. We use a constant step-size tuned during a 10 iteration burn-in period for the non-smooth network. We then use that same step-size for the smoothed counterpart.

d) Smoothing strategy: After optimizing the non-smooth network, we record the maximal norm of the parameters along the iterations of stochastic gradient optimization, that we denote R_t for $t \in \{1, \ldots, \tau\}$. We then smooth the network on the set $C = \bigotimes_{t=1}^\tau B_{R_t}(\mathbb{R}^{p_t})$ such that the resulting objective is a μ-approximation of the original objective for μ varying in $\{10^{-6}, \ldots, 10^1\}$. We then apply stochastic gradient optimization on the smoothed network with the same constant step-size as for the non-smooth network. For both non-smooth and smoothed networks we run the stochastic gradient descent for 200 epochs on the dataset.

B. Results

a) Smoothing effect: We present in Tables I and II the maximal bound on the parameters found after one pass on the network and the resulting smoothing chosen for an accuracy on the objective $\mu = 1$.

b) Performance comparison: We present in Table III the standard deviation on the performance of non-smoothed and smoothed networks. The standard deviation is computed over all possible values of the smoothing for $\mu \in \{10^{-6}, \ldots, 10^1\}$ and over 10 random runs of the stochastic gradient descent. We do not observe any noticeable difference in terms of

	MLP	ConvNet
Train Loss	$1.64 \cdot 10^{-2} \pm 4.39 \cdot 10^{-6}$	$0.57 \pm 1.85 \cdot 10^{-2}$
Test loss	$1.54 \cdot 10^{-2} \pm 4.52 \cdot 10^{-5}$	$2.04 \cdot 10^{-2} \pm 6.21 \cdot 10^{-4}$
Train Acc.	100.0 ± 0	76.31 ± 0.79
Test Acc.	$98.33 \pm 4.20 \cdot 10^{-2}$	72.77 ± 0.76

TABLE III: Average performance of the smoothed counterparts for a range of target accuracy $\mu \in \{10^{-6}, \dots, 10^{1}\}$.

overall performance between the smoothed and non-smoothed networks.

c) Optimization paths: In Fig. 4, we present the norm of the difference between the iterations of stochastic gradient optimization applied to the non-smooth network and the iterations of stochastic gradient optimization applied to the smoothed networks. Here we observe that the trajectory differs for a large smoothing amount. For a target accuracy $\mu = 10$ the difference may then be important. However, for a small target accuracy in the range ($\mu \in \{10^{-6}, \dots, 10^{-3}\}$), the differences in the optimization trajectories become negligible. These plots suggest that a nonsmooth deep neural network can be seen as the limit of a sequence of smoothed counterpart with vanishing amount of smoothing as far as the training process by stochastic optimization is concerned.

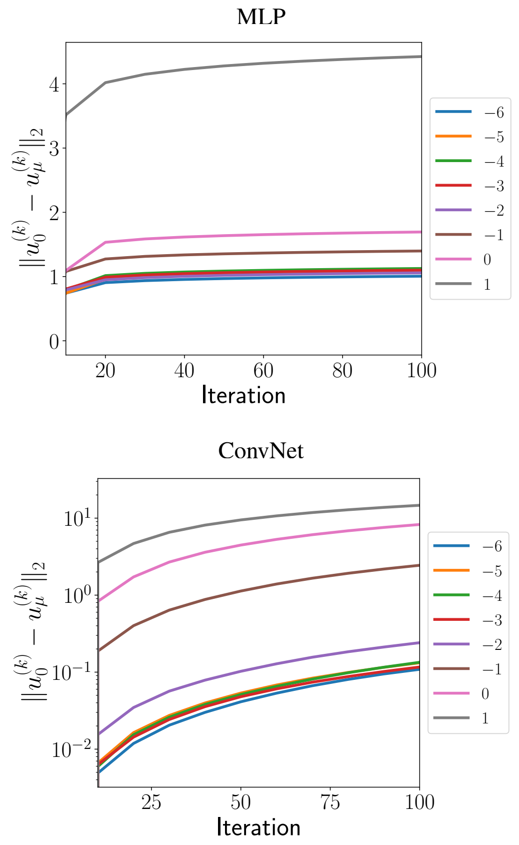

Fig. 4: Divergence of the optimization trajectory of a stochastic gradient descent applied to the smoothed networks $u_\mu^{(k)}$ compared to the non-smoothed one $u_0^{(k)}$ for different values of μ (in $\log_{10}$ scale).

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015.

[2] M. Anthony and P. Bartlett. *Neural Network Learning: Theoretical Foundations*. Cambridge University Press, 2009.

[3] J. Bolte and E. Pauwels. A mathematical model for automatic differentiation in machine learning. *Advances in Neural Information Processing Systems*, 2020.

[4] D. Davis, D. Drusvyatskiy, S. Kakade, and J. D. Lee. Stochastic subgradient method converges on tame functions. *Foundations of computational mathematics*, 20(1):119–154, 2020.

[5] R. Duda, P. Hart, and D. Stork. *Pattern classification*. John Wiley & Sons, 2nd edition, 2012.

[6] S. Eger, P. Youssef, and I. Gurevych. Is it time to swish? comparing deep learning activation functions across NLP tasks. In E. Riloff, D. Chiang, J. Hockenmaier, and J. Tsujii, editors, *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pages 4415–4424. Association for Computational Linguistics, 2018.

[7] S. Ghadimi, G. Lan, and H. Zhang. Mini-batch stochastic approximation methods for nonconvex stochastic composite optimization. *Mathematical Programming*, 155(1-2):267–305, 2016.

[8] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. The MIT Press, 2016.

[9] R. A. Horn and C. R. Johnson. *Matrix analysis*. Cambridge university press, 2nd edition, 2012.

[10] S. M. Kakade and J. D. Lee. Provably correct automatic sub-differentiation for qualified programs. *Advances in neural information processing systems*, 31:7125–7135, 2018.

[11] Y. Lecun. A theoretical framework for back-propagation. In *1988 Connectionist Models Summer School, CMU, Pittsburg, PA*, 1988.

[12] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in PyTorch. 2017.

[13] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[14] S. Shalev-Shwartz and S. Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 2014.

[15] P. Werbos. *The Roots of Backpropagation: From Ordered Derivatives to Neural Networks and Political Forecasting*. Wiley-Interscience, 1994.

Appendix A
Notations

A. Matrices

For a matrix $M \in \mathbb{R}^{d\times n}$, we denote by $\mathrm{Vec}(M)$ the concatenation of the columns of M. We denote $\|M\|_{2,2} = \sup_{x\neq 0, y\neq 0} \frac{x^\top M y}{\|x\|_2\|y\|_2}$ its norm induced by the Euclidean norm and $\|M\|_F = \sqrt{\sum_{ij} M_{ij}^2}$ its Frobenius norm.

B. Tensors

A tensor $\mathcal{A} = (a_{ijk})_{i\in\{1,\ldots,d\}, j\in\{1,\ldots,n\}, k\in\{1,\ldots,p\}} \in \mathbb{R}^{d\times n\times p}$ is represented as a list of matrices $\mathcal{A} = (A_1, \ldots, A_p)$ where $A_k = (a_{ijk})_{i\in\{1,\ldots,d\}, j\in\{1,\ldots,n\}} \in \mathbb{R}^{d\times n}$ for $k \in \{1, \ldots p\}$. Given matrices $P \in \mathbb{R}^{d\times d'}, Q \in \mathbb{R}^{n\times n'}, R \in \mathbb{R}^{p\times p'}$, we denote

$$\mathcal{A}[P,Q,R] = \left(\sum_{k=1}^{p} R_{k,1} P^\top A_k Q, \ldots, \sum_{k=1}^{p} R_{k,p'} P^\top A_k Q\right) \in \mathbb{R}^{d'\times n'\times p'}$$

If P, Q or R are identity matrices, we use the symbol "$\cdot$" in place of the identity matrix. For example, we denote $\mathcal{A}[P,Q,\mathrm{I}_p] = \mathcal{A}[P,Q,\cdot] = \big(P^\top A_1 Q, \ldots, P^\top A_p Q\big)$. If P, Q or R are vectors we consider the flatten object. In particular, for $x \in \mathbb{R}^d, y \in \mathbb{R}^n$, we denote

$$\mathcal{A}[x,y,\cdot] = \begin{pmatrix} x^\top A_1 y \\ \vdots \\ x^\top A_p y \end{pmatrix} \in \mathbb{R}^p,$$

rather than having $\mathcal{A}[x,y,\cdot] \in \mathbb{R}^{1\times 1\times p}$. Similarly, for $z \in \mathbb{R}^p$, we have

$$\mathcal{A}[\cdot,\cdot,z] = \sum_{k=1}^{p} z_k A_k \in \mathbb{R}^{d\times n}.$$

For a tensor $\mathcal{A} = (A_1, \ldots, A_p) \in \mathbb{R}^{d,n,p}$ we denote $\mathcal{A}^t = (A_1^\top, \ldots, A_p^\top) \in \mathbb{R}^{n,d,p}$. We denote the outer product of three vectors $x \in \mathbb{R}^d, y \in \mathbb{R}^n, z \in \mathbb{R}^p$ as $x \boxtimes y \boxtimes z \in \mathbb{R}^{d\times n\times p}$ such that

$$(x \boxtimes y \boxtimes z)_{ijk} = x_i y_j z_k.$$

We define the norm of a tensor $\mathcal{A}$ induced by the Euclidean norm as follows.

Definition 10. *The norm of a tensor $\mathcal{A}$ induced by the Euclidean norm is defined as*

$$\|\mathcal{A}\|_{2,2,2} = \sup_{x\neq 0, y\neq 0, z\neq 0} \frac{\mathcal{A}[x,y,z]}{\|x\|_2\|y\|_2\|z\|_2}. \tag{6}$$

Fact 11. *The tensor norm satisfies the following properties, for a given tensor $\mathcal{A} \in \mathbb{R}^{d\times n\times p}$,*

1) $\|\mathcal{A}\|_{2,2,2} = \|\mathcal{A}^t\|_{2,2,2}$,
2) $\|\mathcal{A}[P,Q,R]\|_{2,2,2} \leq \|\mathcal{A}\|_{2,2,2}\|P\|_{2,2}\|Q\|_{2,2}\|R\|_{2,2}$ *for P, Q, R with appropriate sizes,*
3) $\|\mathcal{A}\|_{2,2,2} = \sup_{z\neq 0} \frac{\|\sum_{k=1}^{p} z_k A_k\|_{2,2}}{\|z\|_2}$.

C. Gradients

For a multivariate function $f : \mathbb{R}^d \mapsto \mathbb{R}^n$, composed of n real functions $f^{(j)}$ for $j \in \{1, \ldots n\}$, we denote $\nabla f(x) = (\nabla f^{(1)}(x), \ldots, \nabla f^{(n)}(x)) \in \mathbb{R}^{d\times n}$, that is the transpose of its Jacobian on x, $\nabla f(x) = (\frac{\partial f^{(j)}}{\partial x_i}(x))_{i\in\{1,\ldots,d\}, j\in\{1,\ldots,n\}} \in \mathbb{R}^{d\times n}$. We represent its 2nd order information by a tensor $\nabla^2 f(x) = (\nabla^2 f^{(1)}(x), \ldots, \nabla^2 f^{(n)}(x)) \in \mathbb{R}^{d\times d\times n}$.

Fact 12. *We have for $f : \mathbb{R}^d \to \mathbb{R}^n$, twice differentiable, and $C \subset \operatorname{dom} f$ convex,*

$$\ell_f^C = \sup_{\substack{x,y\in C \\ x\neq y}} \frac{\|f(x)-f(y)\|_2}{\|x-y\|_2} = \sup_{x\in C} \|\nabla f(x)\|_{2,2}, \quad L_f^C = \sup_{\substack{x,y\in C \\ x\neq y}} \frac{\|\nabla f(x)-\nabla f(y)\|_{2,2}}{\|x-y\|_2} = \sup_{x\in C} \|\nabla^2 f(x)\|_{2,2,2}.$$

where $\|\nabla f(x)\|_{2,2}$ denotes the operator norm of $\nabla f(x)$ and $\|\nabla^2 f(x)\|_{2,2,2}$ denotes the tensor norm of $\nabla^2 f(x)$ both with respect to the Euclidean norm.

Proof. We have for $x, y \in C$,

$$\|f(x)-f(y)\|_2 = \|\int_0^1 \nabla f(x+t(y-x))^\top (y-x)dt\|_2 \leq \sup_{x\in C} \|\nabla f(x)\|_{2,2}\|x-y\|_2,$$

$$\|\nabla f(x)-\nabla f(y)\|_{2,2} = \|\int_0^1 \nabla^2 f(x+t(y-x))[y-x,\cdot,\cdot]dt\|_{2,2} \leq \sup_{x\in C} \|\nabla^2 f(x)\|_{2,2,2}\|x-y\|_2,$$

which gives $\ell_f^C \leq \sup_{x\in C} \|\nabla f(x)\|_{2,2}$ and $L_f^C \leq \sup_{x\in C} \|\nabla^2 f(x)\|_{2,2,2}$. The equalities come from the definitions of the gradient and the Hessian. $\square$

For a real function, $f : \mathbb{R}^d \times \mathbb{R}^p \mapsto \mathbb{R}$, whose value is denoted $f(x,y)$, we decompose its gradient $\nabla f(x,y) \in \mathbb{R}^{d+p}$ on $(x,y) \in \mathbb{R}^d \times \mathbb{R}^p$ as

$$\nabla f(x,y) = \begin{pmatrix} \nabla_x f(x,y) \\ \nabla_y f(x,y) \end{pmatrix} \qquad \text{with} \qquad \nabla_x f(x,y) \in \mathbb{R}^d, \quad \nabla_y f(x,y) \in \mathbb{R}^p.$$

Similarly we decompose its Hessian $\nabla f(x,y) \in \mathbb{R}^{(d+p)\times(d+p)}$ on blocks that correspond to the variables (x,y) as follows

$$\nabla^2 f(x,y) = \begin{pmatrix} \nabla_{xx} f(x,y) & \nabla_{xy} f(x,y) \\ \nabla_{yx} f(x,y) & \nabla_{yy} f(x,y) \end{pmatrix}$$
$$\text{with} \qquad \nabla_{xx} f(x,y) \in \mathbb{R}^{d\times d}, \quad \nabla_{yy} f(x,y) \in \mathbb{R}^{p\times p}, \quad \nabla_{xy} f(x,y) = \nabla_{yx} f(x,y)^\top \in \mathbb{R}^{d\times p}.$$

Given a function $f : \mathbb{R}^{d+p} \mapsto \mathbb{R}^n$ and $(x,y) \in \mathbb{R}^d \times \mathbb{R}^p$, we denote $\nabla_x f(x,y) = (\nabla_x f^{(1)}(x,y), \dots, \nabla_x f^{(n)}(x,y)) \in \mathbb{R}^{d\times n}$ and we define similarly $\nabla_y f(x,y) \in \mathbb{R}^{p\times n}$.

For its second order information we define $\nabla_{xx} f(x,y) = (\nabla_{xx} f^{(1)}(x,y), \dots, \nabla_{xx} f^{(n)}(x,y))$, similarly for $\nabla_{xx} f(x,y)$. Dimension of these definitions are

$$\begin{aligned} \nabla_x f(x,y) &\in \mathbb{R}^{d\times n}, & \nabla_y f(x,y) &\in \mathbb{R}^{p\times n}, \\ \nabla_{xx} f(x,y) &\in \mathbb{R}^{d\times d\times n}, & \nabla_{yy} f(x,y) &\in \mathbb{R}^{p\times p\times n}, \\ \nabla_{xy} f(x,y) &\in \mathbb{R}^{d\times p\times n}, & \nabla_{yx} f(x,y) &\in \mathbb{R}^{p\times d\times n}. \end{aligned}$$

D. Bilinear functions

Definition 13. *A function $\beta : \mathbb{R}^d \times \mathbb{R}^n \to \mathbb{R}^p$ is bilinear if it is represented by a tensor $\mathcal{B} \in \mathbb{R}^{d\times n\times p}$ such that for any $x \in \mathbb{R}^d, y \in \mathbb{R}^n$,*

$$\beta(x,y) = \mathcal{B}[x,y,\cdot].$$

The gradient of a bilinear function β represented by a tensor $\mathcal{B} \in \mathbb{R}^{d\times n\times p}$ at a point x,y is given by

$$\nabla_x \beta(x,y) = \mathcal{B}[\cdot,y,\cdot] \in \mathbb{R}^{d\times p}, \qquad \nabla_y \beta(x,y) = \mathcal{B}[x,\cdot,\cdot] \in \mathbb{R}^{n\times p}. \tag{7}$$

The Hessian of the bilinear function is given

$$\nabla^2_{xx} \beta(x,y) = 0, \quad \nabla^2_{yy} \beta(x,y) = 0, \quad \nabla^2_{xy} \beta(x,y) = \mathcal{B}, \quad \nabla^2_{yx} \beta(x,y) = \mathcal{B}^t. \tag{8}$$

A bilinear function is not Lipschitz continuous as can be seen from Eq. (7). It is smooth w.r.t. the Euclidean norm with a smoothness constant given by the tensor norm of $\mathcal{B}$ as shown in the following proposition.

Lemma 14. *The smoothness of a bilinear function β defined by a tensor $\mathcal{B}$ is upper bounded as $L_\beta \leq \|\mathcal{B}\|_{2,2,2}$.*

Proof. We have

$$\|\nabla^2 \beta(x,y)\|_{2,2,2} = \sup_{z\neq 0} \frac{\|\sum_{k=1}^p z_k \tilde{B}_k\|_{2,2}}{\|z\|_2},$$

where $\nabla^2 \beta(x,y) = (\tilde{B}_1, \dots, \tilde{B}_p)$. We have by Eq. (8) that $\sum_{k=1}^p z_k \tilde{B}_k$ is of the form

$$\sum_{k=1}^p z_k \tilde{B}_k = \begin{pmatrix} 0 & \sum_{k=1}^p z_k B_k \\ \sum_{k=1}^p z_k B_k^\top & 0 \end{pmatrix}$$

where $\mathcal{B} = (B_1, \dots, B_p)$. Therefore we get $\|\sum_{k=1}^p z_k \tilde{B}_k\|_{2,2} = \|\sum_{k=1}^p z_k B_k\|_{2,2}$, see [9, Theorem 7.3.3.]. Therefore

$$\|\nabla^2 \beta(x,y)\|_2 = \sup_{z\neq 0} \frac{\|\sum_{k=1}^p z_k \tilde{B}_k\|_{2,2}}{\|z\|_2} = \sup_{z\neq 0} \frac{\|\sum_{k=1}^p z_k B_k\|_{2,2}}{\|z\|_2} = \|\mathcal{B}\|_{2,2,2}.$$

$\square$

Appendix B
Detailed layer structure

The layer t of a deep network can be described by the following components:

(i) a bi-affine operation such as a matrix multiplication or a convolution, denoted $b_t : \mathbb{R}^{p_t} \times \mathbb{R}^{d_{t-1}} \to \mathbb{R}^{\eta_t}$ and decomposed as

$$b_t(x_{t-1}, u_t) = \beta_t(x_{t-1}, u_t) + \beta_t^u(u_t) + \beta_t^x(x_{t-1}) + \beta_t^0, \tag{9}$$

where β_t is bilinear, β_t^u and β_t^x are linear and β_t^0 is a constant vector,

(ii) an activation function, such as the element-wise application of a non-linear function, denoted $\alpha_t : \mathbb{R}^{\eta_t} \to \mathbb{R}^{\eta_t}$,

(iii) a reduction of dimension, such as a pooling operation, denoted $\pi_t : \mathbb{R}^{\eta_t} \to \mathbb{R}^{d_t}$,

(iv) a normalization of the output, such as batch-normalization, denoted $\nu_t : \mathbb{R}^{d_t} \to \mathbb{R}^{d_t}$.

By concatenating the non-affine operations, i.e., defining $a_t = \nu_t \circ \pi_t \circ \alpha_t$, a layer can be written as

$$\phi_t(x_{t-1}, u_t) = a_t(b_t(x_{t-1}, u_t)). \tag{10}$$

Note that some components may not be included, for example some layers do not include normalization. In the following, we consider the non-linear operation a_t to be an arbitrary composition of functions, i.e., $a_t = a_{t,k_t} \circ \ldots \circ a_{t,1}$. We present common examples of the components of a deep network.

1) Linear operations: In the following, we drop the dependency w.r.t. the layer t and denote by $\tilde{\cdot}$ the quantities characterizing the output. We denote by semi-columns the concatenations of matrices by rows, i.e., for $A \in \mathbb{R}^{d\times n}, B \in \mathbb{R}^{q\times n}$, $(A;B) = (A^\top, B^\top)^\top$.

a) Fully connected layer: A *fully connected* layer taking an input of dimension δ is written

$$\tilde{z} = W^\top z + w^0, \tag{11}$$

where $z \in \mathbb{R}^\delta$ is the input, $W \in \mathbb{R}^{\delta\times\tilde{\delta}}$ are the weights of the layer and $w^0 \in \mathbb{R}^{\tilde{\delta}}$ define the offsets. By vectorizing the parameters and the inputs, a fully connected layer can be written as

$$\begin{aligned} &\tilde{x} = \beta(x,u) + \beta^v(u), \\ \text{where} \qquad &\beta(x,u) = W^\top z \in \mathbb{R}^{\tilde{\delta}},\ \beta^v(u) = w^0, \\ &x = z \in \mathbb{R}^\delta,\ u = \operatorname{Vec}(W; w^0) \in \mathbb{R}^{\tilde{\delta}(\delta+1)}. \end{aligned}$$

b) Convolutional layer: A *convolutional* layer convolves an input (images or signals) of dimension δ denoted $z \in \mathbb{R}^\delta$ with n^f affine filters of size s^f defined by weights $W = (w_1, \ldots, w_{n^f}) \in \mathbb{R}^{s^f\times n^f}$ and offsets $w^0 = (w_1^0, \ldots, w_{n^f}^0) \in \mathbb{R}^{n^f}$ through n^p patches. The kth output of the convolution of the input by the jth filter reads

$$\Xi_{j,k} = w_j^\top \Pi_k z + w_j^0, \tag{12}$$

where $\Pi_k \in \mathbb{R}^{s^f\times\delta}$ extracts a patch of size s^f at a given position of the input z. The output $\tilde{z}$ is then given by the concatenation $\tilde{z}_{k+n^p(j-1)} = \Xi_{j,k}$. By vectorizing the inputs and the outputs, the convolution operation is defined by a set of matrices $(\Pi_k)_{k=1}^{n^p}$ such that

$$\begin{aligned} &\tilde{x} = \beta(x,u) + \beta^v(u), \\ \text{where} \qquad &\beta(x,u) = (w_j^\top \Pi_k z)_{j=1,\ldots,n^f;k=1,\ldots,n^p} \in \mathbb{R}^{n^f n^p},\ \beta^v(u) = w^0 \otimes \mathbf{1}_{n^p}, \\ &x = z \in \mathbb{R}^\delta,\ u = \operatorname{Vec}(W; w^0) \in \mathbb{R}^{(s^f+1)n^f},\ W = (w_1, \ldots, w_{n^f}). \end{aligned}$$

2) Activation functions: We consider element-wise activation functions $\alpha : \mathbb{R}^\eta \to \mathbb{R}^\eta$ such that for a given $x = (x_1, \ldots, x_\eta) \in \mathbb{R}^\eta$,

$$\alpha(x) = (\bar{\alpha}(x_1), \ldots, \bar{\alpha}(x_\eta)), \tag{13}$$

for a given scalar function $\bar{\alpha}$ such as $\bar{\alpha}(x) = \max(x, 0)$ for the Rectified Linear Unit (ReLU) or $\bar{\alpha}(x) = (1+\exp(-x))^{-1}$ for the sigmoid function.

3) Pooling functions: A pooling layer reduces the dimension of the output. For example, an average pooling convolves an input image with a mean filter. Formally, for an input $z \in \mathbb{R}^\delta$, the average pooling with a patch size s^f for inputs with n^f channels and n^p coordinates such that $d = n^f n^p$ convolves the inputs with a filter $P = \mathbf{1}_{s^f} \mathbf{1}_{n^f}^\top / s^f$. The output dimension for each input is $\tilde{\delta} = n^f \tilde{n^p}$ and the patches, represented by some $(\Pi_k)_{k=1}^{\tilde{n^p}}$ acting in Eq. (12), are chosen such that it induces a reduction of dimension, i.e., $\tilde{n^p} \leq n^p$.

4) Normalization functions: Given a batch of input $Z \in \mathbb{R}^{\delta\times m}$ the batch-normalization outputs $\tilde{Z}$ defined by

$$\begin{aligned} &(\tilde{Z})_{ij} = \frac{Z_{ij} - \mu_i}{\sqrt{\epsilon + \sigma_i^2}}, \\ \text{where} \qquad &\mu_i = \frac{1}{m}\sum_{j=1}^m Z_{ij}, \quad \sigma_i^2 = \frac{1}{m}\sum_{j=1}^m (Z_{ij} - \mu_i)^2, \end{aligned} \tag{14}$$

with $\epsilon > 0$, such that the vectorized formulation of the batch-normalization reads $\nu(x) = \operatorname{Vec}(\tilde{Z})$ for $x = \operatorname{Vec}(Z)$.

APPENDIX C
SMOOTHNESS COMPUTATIONS PROOFS

A. Elementary operations

a) Univariate functions:

Lemma 15. *Let $\alpha_i \in \mathcal{C}_{\ell_i, L_i}$ for $i = 1, \ldots, n$. Denote $\ell = (\ell_i)_{i=1}^n$, $L = (L_i)_{i=1}^n$.*

1) *Assume $\alpha_i : \mathbb{R}^{d_i} \to \mathbb{R}^{m_i}$, then*
$$a : \begin{cases} \mathbb{R}^{\sum_{i=1}^n d_i} & \to \mathbb{R}^{\sum_{i=1}^n m_i} \\ x = (x_1; \ldots; x_n) & \to (\alpha_1(x_1); \ldots; \alpha_n(x_n)) \end{cases}$$
is $\|\ell\|_2$-Lipschitz continuous and $\|L\|_\infty$-smooth.

2) *Assume $\alpha_i : \mathbb{R}^{d_i} \to \mathbb{R}^{m}$, then*
$$a : \begin{cases} \mathbb{R}^{\sum_{i=1}^n d_i} & \to \mathbb{R}^{m} \\ x = (x_1; \ldots; x_n) & \to \sum_{i=1}^n \alpha_i(x_i) \end{cases}$$
is $\|\ell\|_2$-Lipschitz continuous and $\|L\|_\infty$-smooth.

3) *Assume $a_i : \mathbb{R}^{d} \to \mathbb{R}^{m_i}$, then*
$$a : \begin{cases} \mathbb{R}^{d} & \to \mathbb{R}^{\sum_{i=1}^n m_i} \\ x & \to (\alpha_1(x); \ldots; \alpha_n(x)) \end{cases}$$
is $\|\ell\|_2$-Lipschitz continuous and $\|L\|_2$-smooth.

4) *Assume $\alpha_i : \mathbb{R}^{d} \to \mathbb{R}^{m}$, then*
$$a : \begin{cases} \mathbb{R}^{d} & \to \mathbb{R}^{m} \\ x & \to \sum_{i=1}^n \alpha_i(x) \end{cases}$$
is $\|\ell\|_1$-Lipschitz continuous and $\|L\|_1$-smooth.

Proof.

1) We have for $x = (x_1; \ldots; x_n) \in \mathbb{R}^{\sum_{i=1}^n d_i}$ and $z = (z_1; \ldots; z_n) \in \mathbb{R}^{\sum_{i=1}^n m_i}$,
$$\|\nabla a(x) z\|_2 = \|\sum_{i=1}^n \nabla \alpha_i(x_i) z_i\|_2 \leq \sum_{i=1}^n \|z_i\|_2 \|\nabla \alpha_i(x_i)\|_{2,2} \leq \|z\|_2 \sqrt{\sum_{i=1}^n \ell_i^2},$$
which gives an upper bound on the Lipschitz-continuity of a. For $x = (x_1; \ldots; x_n), y = (y_1; \ldots; y_n) \in \mathbb{R}^{\sum_{i=1}^n d_i}$, we have
$$\|(\nabla a(x) - \nabla a(y)) z\|_2 \leq \sum_{i=1}^n \|z_i\|_2 \|\nabla \alpha_i(x_i) - \nabla \alpha_i(y_i)\|_{2,2} \leq \sum_{i=1}^n \|z_i\|_2 \|x_i - y_i\|_2 L_i \leq \|z\|_2 \|x - y\|_2 \max_{i \in \{1, \ldots, n\}} L_i.$$
Hence $\|\nabla a(x) - \nabla a(y)\|_{2,2} \leq \|x - y\|_2 \max_{i \in \{1, \ldots, n\}} L_i$ which gives an upper bound on the smoothness of a.

2) We have for $x = (x_1; \ldots; x_n) \in \mathbb{R}^{\sum_{i=1}^n d_i}$ and $z \in \mathbb{R}^m$,
$$\|\nabla a(x) z\|_2^2 = \sum_{i=1}^n \|\nabla \alpha_i(x_i) z\|_2^2 \leq \sum_{i=1}^n \ell_i^2 \|z\|_2^2,$$
which gives the Lipschitz-continuity parameter. Similarly we have for $x = (x_1; \ldots; x_n), y = (y_1; \ldots; y_n) \in \mathbb{R}^{\sum_{i=1}^n d_i}$,
$$\|(\nabla a(x) - \nabla a(y)) z\|_2^2 = \sum_{i=1}^n \|(\nabla \alpha_i(x_i) - \nabla \alpha_i(y_i)) z\|_2^2 \leq \sum_{i=1}^n L_i^2 \|x_i - y_i\|_2^2 \|z\|_2^2 \leq \max_{i \in \{1, \ldots, n\}} L_i^2 \|x - y\|_2^2 \|z\|_2^2,$$
which gives the smoothness constant of a.

3) The bound on the Lipschitz-continuity parameter follows from the same argument as in 1. For the smoothness parameter, we have for $x, y \in \mathbb{R}^d$ and $z = (z_1; \ldots; z_n) \in \mathbb{R}^{\sum_{i=1}^n m_i}$,
$$\|(\nabla a(x) - \nabla a(y)) z\|_2 \leq \sum_{i=1}^n \|z_i\|_2 \|\nabla \alpha_i(x) - \nabla \alpha_i(y)\|_{2,2} \leq \sum_{i=1}^n \|z_i\|_2 \|x - y\|_2 L_i \leq \|z\|_2 \|x - y\|_2 \sqrt{\sum_{i=1}^n L_i^2}.$$
Hence the result as in 1.

4) Clear by linearity of the gradient and triangular inequality.

□

b) Examples: Consider $y \in \{0,1\}^k$, the logistic loss is defined as $f(\hat{y}) = \mathcal{L}_{\log}(\hat{y}, y) = -y^\top \hat{y} + \log\left(\sum_{j=1}^k \exp(\hat{y}_j)\right)$. We have then, denoting $\exp(y) = (\exp(y_i))_{i=1,\ldots k}$,

$$\nabla f(\hat{y}) = -y + \frac{\exp(\hat{y})}{\exp(\hat{y})^\top \mathbf{1}_k}, \qquad \nabla^2 f(\hat{y}) = \frac{\mathbf{diag}(\exp(\hat{y}))}{\exp(\hat{y})^\top \mathbf{1}_k} - \frac{\exp(\hat{y})\exp(\hat{y})^\top}{(\exp(\hat{y})^\top \mathbf{1}_k)^2}.$$

Therefore using that $y \in \{0,1\}^k$ and that $\|\exp(\hat{y})\|_2 \leq \|\exp(\hat{y})\|_1$,

$$\ell_{\log} \leq 2, \qquad L_{\log} \leq 2.$$

c) Bilinear functions:

Lemma 16. *Consider $s \times t$ bilinear functions $\beta_{i+(j-1)s} : \mathbb{R}^{d_i} \times \mathbb{R}^{p_j} \to \mathbb{R}^{m_{i+(j-1)s}}$ for $i \in \{1, \ldots s\}, j \in \{1, \ldots t\}$ then*

$$\beta : \begin{cases} \mathbb{R}^{\sum_{i=1}^s d_i} \times \mathbb{R}^{\sum_{j=1}^t p_j} & \to \mathbb{R}^{\sum_{k=1}^{st} m_k} \\ (x,u) & \to (\beta_1(x_1,u_1); \ldots; \beta_s(x_s,u_1); \beta_{s+1}(x_1,u_2); \ldots; \beta_{st}(x_s,u_t)) \end{cases}$$

is $L_\beta = \max_{k \in \{1,\ldots,st\}} L_{\beta_k}$ smooth.

Proof. By Lemma 14, we have that $L_\beta = \sup_{x,u} \|\beta(x,u)\|_2 / \|x\|_2 \|u\|_2$. Now $\|\beta(x,u)\|_2^2 = \sum_{i=1}^s \sum_{j=1}^t \|\beta_{i+s(j-1)}(x_i,u_j)\|_2^2 \leq \sum_{i=1}^s \sum_{j=1}^t L_{\beta_{i+s(j-1)}} \|x_i\|_2^2 \|u_j\|_2^2 \leq \max_{k\in\{1,\ldots,st\}} L_{\beta_k} \|x\|_2^2 \|u\|_2^2$. □

d) Examples: For a fully connected layer, the bilinear function $\beta(x,u) \to U^\top X$ for $u = \mathrm{Vec}(U)$, $x = \mathrm{Vec}(X)$ is clearly 1-smooth (because $\|U^\top X\|_F \leq \|U\|_F \|X\|_F$). The linear part β^u is clearly 1-Lipschitz continuous. So we get

$$L_{full} = 1, \quad \beta^u_{full} = 1.$$

For a convolution, by Lemma 16, we only need to compute the smoothness of the convolution of an image with one filter. This is done by the following Lemma.

Lemma 17. *Consider p subsets S_k of $\{1, \ldots, n\}$ of size $|S_k| = d$. Denote $\Pi_k \in \{0,1\}^{d \times n}$ the linear form that extracts the S_k coordinates of a vector of size n, i.e., $\Pi_k z = z_{S_k}$ for $z \in \mathbb{R}^n$. The convolution of $z \in \mathbb{R}^n$ by $w \in \mathbb{R}^d$ through the p subsets S_k defined as*

$$\beta(z,w) = (w^\top \Pi_1 z; \ldots; w^\top \Pi_p z)$$

is $L_\beta = \sqrt{\max_{i=1,\ldots,n} |V_i|}$-smooth where $V_i = \{S_j : i \in S_j\}$.

Proof. We have $\|\beta(z,w)\|_2^2 = \sum_{j=1}^p (w^\top \Pi_j z)^2 \leq \sum_{j=1}^p \|w\|_2^2 \|z_{S_j}\|_2^2 = \|w\|_2^2 \sum_{i=1}^d \sum_{S_j \in V_i} z_i^2 \leq \|w\|_2^2 \max_{i=1,\ldots,n} |V_i| \|z\|_2^2$. □

Concretely for a convolution such that at most p patches contain a coordinate i the convolution is $\sqrt{p}$-smooth. If the patches do not overlap then the convolution is 1-smooth. If the convolution has a stride of 1 and the operation is normalized by the size of the filters then the convolution has again a smoothness constant of 1. Generally for a 2d convolution with a kernel of size $k \times k$ and a stride of s, we have

$$L_{conv} = \left\lceil \frac{k}{s} \right\rceil^2, \quad \ell_{conv} = 1.$$

e) Batch of inputs: For batch of inputs, the smoothness constants of the non-linear and bilinear parts do not change by Lemmas 15 and 16. The Lipschitz-constant of the lienar part of the biaffine function is modified using Lemma 15 item 3. Namely for a batch of size m, the fully connected layers or the convolutional layers have a linear part whose Lipschitz constant is given by $l_b = \sqrt{m}$.

B. Compositions

The smoothness properties of the functions can be derived by bounding appropriately their first and second order information. Even if e.g. the functions are not twice differentiable, the same results would apply by decomposing carefully the terms, we directly use the second order information as it directly gives what we are interested in.

For a smooth function, an upper bound on the Lipschitz continuity of the function on a bounded set can be estimated even if the function is not Lipschitz continuous. Similarly a bound on the function a bounded set can be refined as defined below.

Fact 18. *For a function $f \in \mathcal{C}_{m_f, \ell_f, L_f}$ and $R > 0$. Denoting $B_R = \{x \in \mathrm{dom}\, f : \|x\|_2 \leq R\}$, we have that*

$$\ell_f^{B_R} \leq \ell_f(R) := \min\{\ell_f, \|\nabla f(0)\|_{2,2} + R L_f\},$$
$$m_f^{B_R} \leq m_f(R) := \min\{m_f, \|f(0)\|_2 + R\ell_f(R)\}$$

For a sequence of compositions we have the following result.

Lemma 19. *Consider*

$$a = a_k \circ \ldots \circ a_1$$

with $a_j \in \mathcal{C}_{m_{a_j}, \ell_{a_j}, L_{a_j}}$ for $j \in \{1, \ldots, k\}$ and $a : \mathbb{R}^d \to \mathbb{R}^n$. Denote $B_R = \{x \in \mathbb{R}^d : \|x\|_2 \leq R\}$, and for $j \in \{1, \ldots, k\}$,

$$\begin{aligned} m_j &= m_{a_j}(m_{t-1}), \\ \ell_j &= \ell_{j-1}\ell_{a_j}(m_{j-1}), \\ L_j &= L_{a_j}\ell_{j-1}^2 + L_{j-1}\ell_{a_j}(m_{j-1}), \end{aligned}$$

with $m_0 = R, \ell_0 = 1, L_0 = 0$. We have

$$m_a^{B_R} \leq m_\tau, \qquad \ell_a^{B_R} \leq \ell_\tau = \prod_{j=1}^{k} \ell_{a_j}(m_{j-1}), \qquad L_a^{B_R} \leq L_\tau = \sum_{j=1}^{k} L_{a_j} \left(\prod_{i=1}^{j-1} \ell_{a_i}(m_{i-1})\right)^2 \left(\prod_{i=j+1}^{k} \ell_{a_i}(m_{i-1})\right).$$

Proof. The bound on the output is a direct iterative application of Fact 18. We have for $x \in \mathbb{R}^d$,

$$\nabla a(x) = \prod_{j=1}^{k} g_j(x), \qquad \text{where} \quad g_j(x) = \nabla a_j(a_{j-1} \circ \ldots \circ a_1(x)) \quad \text{for } j \in \{1, \ldots, k\}.$$

We have

$$\sup_{x \in \mathbb{R}^d : \|x\|_2 \leq R} \|g_j(x)\|_{2,2} \leq \min\{\ell_{a_j}, \|\nabla a_j(0)\|_{2,2} + L_{a_j} m_{a_{j-1} \circ \ldots \circ a_1}^{B_R}\}.$$

Therefore

$$\ell_a^{B_R} \leq \prod_{j=1}^{k} \ell_{a_j}(m_{j-1}).$$

We have for $x \in \mathbb{R}^d$,

$$\nabla^2 a(x) = \sum_{j=1}^{k} \nabla^2 a_j(x) \left[\left(\prod_{i=1}^{j-1} g_i(x)\right)^\top, \left(\prod_{i=1}^{j-1} g_i(x)\right)^\top, \prod_{i=j+1}^{k} g_i(x) \right].$$

Therefore

$$L_a^{B_R} \leq \sum_{j=1}^{k} L_{a_j} \left(\prod_{i=1}^{j-1} \ell_{a_i}(m_{i-1})\right)^2 \left(\prod_{i=j+1}^{k} \ell_{a_i}(m_{i-1})\right).$$

□

Lemma 19 can be used to estimate the smoothness of a chain of computations with respect to its input for fixed parameters.

Corollary 20. *Consider a chain f of τ computations $\phi_t \in \mathcal{C}_{m_{\phi_t}, \ell_{\phi_t}, L_{\phi_t}}$ with given parameters $u = (u_1; \ldots; u_\tau)$. Denote $\phi_{t,u_t} = \phi_t(\cdot, u_t)$. Denote $B_R = \{x \in \mathbb{R}^d : \|x\|_2 \leq R\}$, and for $j \in \{1, \ldots, k\}$,*

$$\begin{aligned} m_j &= m_{\phi_t(\cdot, u_t)}(m_{t-1}), \\ \ell_j &= \ell_{j-1}\ell_{\phi_j(\cdot, u_j)}(m_{j-1}), \\ L_j &= L_{\phi_j(\cdot, u_j)}\ell_{j-1}^2 + L_{j-1}\ell_{\phi_j(\cdot, u_j)}(m_{j-1}), \end{aligned}$$

with $m_0 = R, \ell_0 = 1, L_0 = 0$. We have

$$m_{f_{\tau,u}}^{B_R} \leq m_\tau, \quad \ell_{f_{\tau,u}}^{B_R} \leq \ell_\tau = \prod_{j=1}^{k} \ell_{\phi_j(\cdot, u_j)}(m_{j-1}), \quad L_{f_{\tau,u}}^{B_R} \leq L_\tau = \sum_{j=1}^{k} L_{\phi_j(\cdot, u_j)} \left(\prod_{i=1}^{j-1} \ell_{\phi_i(\cdot, u_i)}(m_{i-1})\right)^2 \left(\prod_{i=j+1}^{k} \ell_{\phi_i(\cdot, u_i)}(m_{i-1})\right).$$

C. Chains of computations

Lemma 21. *Let f be a chain of τ computations ϕ_t. Let $u = (u_1; \ldots; u_\tau) \in \mathbb{R}^{\sum_{t=1}^{\tau} p_t}$, $x_0 \in \mathbb{R}^{d_0}$, denote $x_t = f_t(x_0, u)$ and $E_t^\top = (0_{p_t d_0}, 0_{p_t p_1}, \ldots, \mathrm{I}_{p_t p_t}, \ldots, 0_{p_t p_\tau}) \in \mathbb{R}^{p_t \times (d_0 + \sum_{t=1}^{\tau} p_t)}$ such that $E_t^\top (x_0; u) = u_t$ for $t \in \{1, \ldots, \tau\}$.*

1) If ϕ_t are differentiable, then

$$\nabla f_t(x_0, u) = \nabla f_{t-1}(x_0, u) \nabla_{x_{t-1}} \phi_t(x_{t-1}, u_t) + E_t \nabla_{u_t} \phi_t(x_{t-1}, u_t)$$

with $\nabla f_0(x_0, u) = E_0$, with $E_0^\top = (\mathrm{I}_{d_0 d_0}, 0_{d_0 p_1}, \ldots, \ldots, 0_{d_0 p_\tau}) \in \mathbb{R}^{d_0 \times (d_0 + \sum_{t=1}^{\tau} p_t)}$ such that $E_0^\top (x_0; u) = x_0$.

2) If ϕ_t are twice differentiable,

$$\begin{aligned}\nabla^2 f_t(x_0,u) = \nabla^2 f_{t-1}(x_0,u)[\cdot,\cdot,\nabla_{x_{t-1}}\phi_t(u_t,x_{t-1})] \\ + \nabla^2_{x_{t-1}x_{t-1}}\phi_t(x_{t-1},u_t)[\nabla f_{t-1}(x_0,u)^\top,\nabla f_{t-1}(x_0,u)^\top,\cdot] \\ + \nabla^2_{x_{t-1}u_t}\phi_t(x_{t-1},u_t)[\nabla f_{t-1}(x_0,u)^\top,E_t^\top,\cdot] \\ + \nabla^2_{u_t x_{t-1}}\phi_t(x_{t-1},u_t)[E_t^\top,\nabla f_{t-1}(x_0,u)^\top,\cdot] \\ + \nabla^2_{u_t u_t}\phi_t(x_{t-1},u_t)[E_t^\top,E_t^\top,\cdot]\end{aligned} \tag{15}$$

with $\nabla^2 f_0(x_0,u)=0$.

Proof. It follows from the definition of the chain of computations and the notations used for tensors. Precisely we have that

$$f_t(x_0,u)=\phi_t(f_{t-1}(x_0,u),E_t^\top(x_0;u)),$$

hence the first result that can be written

$$\nabla f_t(x_0,u)=\nabla f_{t-1}(x_0,u)\nabla_{x_{t-1}}\phi_t(f_{t-1}(x_0,u),E_t^\top(x_0;u))+E_t\nabla_{u_t}\phi_t(f_{t-1}(x_0,u),E_t^\top(x_0;u)),$$

hence the second result using the tensor notations. □

We have the following result for smooth and Lipschitz continuous chains of computations.

Lemma 2. *Consider a chain f of τ computations $\phi_t\in\mathcal{C}_{\ell_{\phi_t},L_{\phi_t}}$ initialized at some $x_0\in\mathbb{R}^{d_0}$.*
(i) We have $\ell_{f_{\tau,x_0}}\le\ell_\tau$, where

$$\ell_0{=}0,\ \ \ell_t{=}\ell_{\phi_t}+\ell_{t-1}\ell_{\phi_t},\ \ \textit{for } t\in\{1,\dots,\tau\}.$$

(ii) We have $L_{f_{\tau,x_0}}\le L_\tau$, where

$$L_0{=}0,\ \ L_t{=}L_{t-1}\ell_{\phi_t}+L_{\phi_t}(1+\ell_{t-1})^2,\ \ \textit{for } t\in\{1,\dots,\tau\}.$$

Proof. The first claim follows directly from Lemma 21. For the second claim we have that (15) gives

$$L_{f_t}\le L_{f_{t-1}}\ell_{\phi_t}+L_{\phi_t}\ell_{f_{t-1}}^2+2L_{\phi_t}\ell_{f_{t-1}}+L_{\phi_t},$$

which simplifies to give the result. □

For a bivariate function $\phi(x,u):\mathbb{R}^d\times\mathbb{R}^p\to\mathbb{R}^\eta$, we define

$$\ell_\phi^u=\sup_{u\in\mathbb{R}^p,x\in\mathbb{R}^d}\ell_{\phi(x,u+\cdot)},\qquad \ell_\phi^x=\sup_{u\in\mathbb{R}^p,x\in\mathbb{R}^d}\ell_{\phi(x+\cdot,u)}.$$

Moreover if the function is continuously differentiable, we define

$$\begin{aligned}L_\phi^{uu}&=\sup_{u\in\mathbb{R}^p,x\in\mathbb{R}^d}\ell_{\nabla_u\phi(x,u+\cdot)}, & L_\phi^{xu}&=\sup_{u\in\mathbb{R}^p,x\in\mathbb{R}^d}\ell_{\nabla_u\phi(x+\cdot,u)},\\ L_\phi^{ux}&=\sup_{u\in\mathbb{R}^p,x\in\mathbb{R}^d}\ell_{\nabla_x\phi(x,u+\cdot)}, & L_\phi^{xx}&=\sup_{u\in\mathbb{R}^p,x\in\mathbb{R}^d}\ell_{\nabla_x\phi(x+\cdot,u)}.\end{aligned}$$

For a bivariate continuosly differentiable function $\phi(x,u):\mathbb{R}^p\times\mathbb{R}^d\to\mathbb{R}^\eta$, we have that

$$\ell_\phi^u=\sup_{u\in\mathbb{R}^p,x\in\mathbb{R}^d}\|\nabla_u\phi(x,u)\|_{2,2},\qquad \ell_\phi^x=\sup_{u\in\mathbb{R}^p,x\in\mathbb{R}^d}\|\nabla_x\phi(x,u)\|_{2,2}.$$

If the function ϕ is twice continuously differentiable, we have that

$$\begin{aligned}L_\phi^{uu}&=\sup_{u\in\mathbb{R}^p,x\in\mathbb{R}^d}\|\nabla^2_{uu}\phi(x,u)\|_{2,2,2}, & L_\phi^{xu}&=\sup_{u\in\mathbb{R}^p,x\in\mathbb{R}^d}\|\nabla^2_{xu}\phi(x,u)\|_{2,2,2},\\ L_\phi^{ux}&=\sup_{u\in\mathbb{R}^p,x\in\mathbb{R}^d}\|\nabla^2_{ux}\phi(x,u)\|_{2,2,2}, & L_\phi^{xx}&=\sup_{u\in\mathbb{R}^p,x\in\mathbb{R}^d}\|\nabla^2_{uu}\phi(x,u)\|_{2,2,2}.\end{aligned}$$

Finally for $R_x\ge 0, R_u\ge 0$, we have

$$\begin{aligned}\sup_{(x,u)\in B_{R_x}\times B_{R_u}}\|\nabla_u\phi(x,u)\|_{2,2}\le\ell_\phi^u(R_x,R_u):=\min\{\ell_\phi^u,\|\nabla_u\phi(0,0)\|_{2,2}+L_\phi^{uu}R_u+L_\phi^{xu}R_x\}\\ \sup_{(x,u)\in B_{R_x}\times B_{R_u}}\|\nabla_x\phi(x,u)\|_{2,2}\le\ell_\phi^x(R_x,R_u):=\min\{\ell_\phi^x,\|\nabla_x\phi(0,0)\|_{2,2}+L_\phi^{xx}R_u+L_\phi^{ux}R_x\}.\end{aligned} \tag{16}$$

We then have the following.

Lemma 22. *Let f be a chain of τ computations $\phi_t \in \mathcal{C}_{m_{\phi_t}, \ell^u_{\phi_t}, \ell^x_{\phi_t}, L^{uu}_{\phi_t}, L^{xu}_{\phi_t}, L^{xx}_{\phi_t}}$, initialized at some x_0 such that $\|x_0\|_2 \leq R_0$. Let $C = \bigotimes_{t=1}^{\tau} B_{R_t}(\mathbb{R}^{p_t}) = \{u = (u_1; \ldots; u_\tau) \in \mathbb{R}^{\sum_{t=1}^{\tau} p_t} : u_t \in \mathbb{R}^{p_t}, \|u_t\|_2 \leq R_t\}$. Define for $t \in \{1, \ldots, \tau\}$,*

$$\begin{aligned} m_t &= \min\{m_{\phi_t}, \|\phi_t(0,0)\|_2 + \ell^u_{\phi_t}(m_{t-1}, R_t)R_t + \ell^x_{\phi_t}(m_{t-1}, R_t)m_{t-1}\}, \\ \ell_t &= \ell^u_{\phi_t}(m_{t-1}, R_t) + \ell_{t-1}\ell^x_{\phi_t}(m_{t-1}, R_t), \\ L_t &= L_{t-1}\ell^x_{\phi_t}(m_{t-1}, R_t) + L^{xx}_{\phi_t}\ell^2_{t-1} + (L^{xu}_{\phi_t} + L^{ux}_{\phi_t})\ell_{t-1} + L^{uu}_{\phi_t}. \end{aligned}$$

with $m_0 = R_0$, $\ell_0 = 0$, $L_0 = 0$. We have that

$$m^C_{f_\tau, x_0} \leq m_\tau, \quad \ell^C_{f_\tau, x_0} \leq \ell_\tau, \quad L^C_{f_\tau, x_0} \leq L_\tau.$$

Proof. The result directly follows from Lemma 21, with the Lipschitz-continuity constants derived in (16). □

Proposition 4. *Consider a chain f of τ computations whose layers ϕ_t are defined by*

$$\phi_t(x_{t-1}, u_t) = a_t(b_t(x_{t-1}, u_t)),$$

for $t \in \{1, \ldots, \tau\}$, where $b_t \in \mathcal{B}_{L_{b_t}, l^u_{b_t}, l^x_{b_t}}$, and a_t is decomposed as

$$a_t = a_{t,k_t} \circ \ldots \circ a_{t,1},$$

with $a_{t,i} \in \mathcal{C}_{m_{a_{t,i}}, \ell_{a_{t,i}}, L_{a_{t,i}}}$. Consider $C = \bigotimes_{t=1}^{\tau} B_{R_t}(\mathbb{R}^{p_t})$. The outputs m_τ, ℓ_τ and L_τ of Algo. 1 satisfy $m^C_{f_\tau, x_0} \leq m_\tau$, $\ell^C_{f_\tau, x_0} \leq \ell_\tau$, $L^C_{f_\tau, x_0} \leq L_\tau$.

Proof. The proof relies on Lemma 22, where the smoothness of the inner compositions are computed according to Lemma 19. Namely, we have

$$\ell^u_{\phi_t}(R_x, R_u) \leq \ell_{a_t}(m_{b_t}(R_x, R_u))\ell^u_{b_t}(R_x, R_u), \qquad \ell^x_{\phi_t}(R_x, R_u) \leq \ell_{a_t}(m_{b_t}(R_x, R_u))\ell^x_{b_t}(R_x, R_u),$$

with

$$\begin{aligned} \ell^u_{b_t}(R_x, R_u) = L_{b_t}R_x + l^u_{b_t}, \qquad \ell^x_{b_t}(R_x, R_u) = L_{b_t}R_u + l^x_{b_t}, \\ m_{b_t}(R_x, R_u) = \ell^u_{b_t}(R_x, R_u)R_u + \ell^x_{b_t}(R_x, R_u)R_x + \|b_t(0,0)\|_2, \end{aligned}$$

and ℓ_{a_t} can be computed as in Lemma. 19. On the other hand, denoting $L^{xx}_{\phi_t}(R_x, R_u) = \sup_{(x,u) \in B_{R_x} \times B_{R_u}} \|\nabla^2_{xx}\phi_t(x,u)\|_{2,2,2}$ (and similarly for $L^{uu}_{\phi_t}, L^{ux}_{\phi_t}, L^{xu}_{\phi_t}$), we have

$$\begin{aligned} L^{xx}_{\phi_t}(R_x, R_u) &\leq L_{a_t}(m_{b_t}(R_x, R_u))\ell^x_{b_t}(R_x, R_u)^2 \\ L^{uu}_{\phi_t}(R_x, R_u) &\leq L_{a_t}(m_{b_t}(R_x, R_u))\ell^u_{b_t}(R_x, R_u)^2 \\ L^{xu}(R_x, R_u) = L^{ux}(R_x, R_u) &= L_{b_t}\ell_{a_t}(m_{b_t}(R_x, R_u)) + L_{a_t}(m_{b_t}(R_x, R_u))\ell^u_{b_t}(R_x, R_u)\ell^x_{b_t}(R_x, R_u), \end{aligned}$$

where $L_{a_t}(m_{b_t}(R_x, R_u))$ is computed by Lemma 19. □

Appendix D
Smoothing proofs

A. Elementary operations

The following lemmas provide the smoothing of elementary operations applied to smoothable functions.

Fact 23. *Let $a \in \mathcal{S}_{\ell, L, K}$ and $\lambda \in \mathbb{R}, \lambda \neq 0$. Then $\lambda a \in \mathcal{S}_{|\lambda|\ell, |\lambda|L, \lambda^2 K}$ by taking for $\mu > 0$, $(\lambda a)_\mu = \lambda a_{\mu/\lambda}$.*

Lemma 24. *Let $\alpha_i \in \mathcal{S}_{L_i, K_i}$ for $i = 1, \ldots, n$ and $I = \{j \in \{1, \ldots, k\} : K_j > 0\}$. Denote $L = (L_i)_{i=1}^n$, $K = (K_i)_{i=1}^n$ and define*

$$\nu_i = \begin{cases} \mu/\sqrt{|I|} & \text{if } i \in I \\ 0 & \text{if } i \notin I \end{cases}, \qquad \lambda_i = \begin{cases} \mu/|I| & \text{if } i \in I \\ 0 & \text{if } i \notin I \end{cases}.$$

1) *Assume $\alpha_i : \mathbb{R}^{d_i} \to \mathbb{R}^{m_i}$, then*

$$a : \begin{cases} \mathbb{R}^{\sum_{i=1}^n d_i} & \to \mathbb{R}^{\sum_{i=1}^n m_i} \\ x = (x_1; \ldots; x_n) & \to (\alpha_1(x_1); \ldots; \alpha_n(x_n)) \end{cases}$$

is $(\|L\|_\infty, \sqrt{|I|}\|K\|_\infty)$-smoothable by defining for $\mu > 0$, $a_\mu(x) = (\alpha_{1,\nu_1}(x_1); \ldots; \alpha_{n,\nu_n}(x_n))$.

2) *Assume $\alpha_i : \mathbb{R}^{d_i} \to \mathbb{R}^m$, then the element-wise sum*

$$a : \begin{cases} \mathbb{R}^{\sum_{i=1}^n d_i} & \to \mathbb{R}^m \\ x = (x_1; \ldots; x_n) & \to \sum_{i=1}^n \alpha_i(x_i) \end{cases}$$

is $(\|L\|_\infty, |I|\|K\|_\infty)$-smoothable by defining for $\mu > 0$, $a_\mu(x) = \sum_{i=1}^n \alpha_{i,\lambda_i}(x_i)$.

3) *Assume $a_i : \mathbb{R}^d \to \mathbb{R}^{m_i}$, then the concatenation of the α_i, defined as*
$$a : \begin{cases} \mathbb{R}^d & \to \mathbb{R}^{\sum_{i=1}^n m_i} \\ x & \to (\alpha_1(x); \ldots ; \alpha_n(x)) \end{cases}$$
is $(\|L\|_2, \sqrt{|I|}\|K\|_2)$-smoothable by defining for $\mu > 0$, $a_\mu(x) = (\alpha_{1,\nu_1}(x); \ldots ; \alpha_{n,\nu_n}(x))$.

4) *Assume $\alpha_i : \mathbb{R}^d \to \mathbb{R}^m$, then*
$$a : \begin{cases} \mathbb{R}^d & \to \mathbb{R}^m \\ x & \to \sum_{i=1}^n \alpha_i(x) \end{cases}$$
is $(\|L\|_1, |I|\|K\|_1)$-smoothable by defining for $\mu > 0$, $a_\mu(x) = \sum_{i=1}^n \alpha_{i,\lambda_i}(x)$.

Proof. 1) For $\mu > 0$, define $a_\mu = (\alpha_{1,\nu_1}(x_1); \ldots ; \alpha_{n,\nu_n}(x_n))$ as proposed in the statement. We have for any $x = (x_1; \ldots ; x_n) \in \mathbb{R}^{\sum_{i=1}^n d_i}$, $\|a(x) - a_\nu(x)\|_2^2 = \sum_{i=1}^n \|\alpha_i(x_i) - \alpha_{\nu_i,i}(x_i)\|_2^2 \leq |I|\nu^2 = \mu$. Hence, a_ν approximates a up to μ precision. The smoothness of α_ν is given by Lemma 15, namely it is $\max_{i\in\{1,\ldots,n\}} L_i + K_i/\nu_i \leq \max_{i\in\{1,\ldots,n\}} L_i + \sqrt{|I|} \max_{i\in\{1,\ldots,n\}} K_i/\mu$.

2) In this case we have $\|a(x) - a_\nu(x)\|_2 \leq \sum_{i=1}^n \|\alpha_i(x_i) - \alpha_{\nu_i,i}(x_i)\|_2 \leq |I|\nu = \mu$. The smoothness computations follow from Lemma 15.

3) The third case can be treated by the same argument as in the first case.

4) Clear.

□

B. Compositions

Lemma 25. *If $a_j \in \mathcal{S}_{\ell_j,L_j,K_j}$ for $j \in \{1, \ldots , k\}$, then $a = a_k \circ \ldots \circ a_1 \in \mathcal{S}_{\ell,L,K}$ with*
$$\ell = \prod_{j=1}^k \ell_j, \qquad L = \sum_{j=1}^k L_j \left(\prod_{i=1}^{j-1} \ell_i\right)^2 \left(\prod_{i=j+1}^k \ell_i\right), \qquad K = |I|\ell^2 \sum_{j\in I} \frac{K_j}{\ell_j^2}$$
where $I = \{j \in \{1, \ldots , k\} : K_j > 0\}$. The smoothing of a to accuracy μ is given by $a_\mu(x) = a_{k,\mu_k} \circ \ldots \circ a_{1,\mu_1}$ with
$$\mu_j = \begin{cases} \mu/(|I| \prod_{i=j+1}^k \ell_i) & \text{if } j \in I \\ 0 & \text{if } j \notin I \end{cases}.$$

Proof. Let $\nu_j > 0$ for $j \in \{1, \ldots k\}$, and consider
$$a_{\nu_{1:k}} = a_{k,\nu_k} \circ \ldots \circ a_{1,\nu_1}.$$
We have for any $x \in \operatorname{dom} a_1$,
$$\begin{aligned} \|a_{\nu_{1:k}}(x) - a(x)\|_2 &\leq \|a_{k,\nu_k} \circ a_{k-1,\nu_{k-1}} \ldots \circ a_{1,\nu_1}(x) - a_{k,\nu_k} \circ a_{k-1} \ldots \circ a_1(x)\|_2 \\ &\quad + \|a_{k,\nu_k} \circ a_{k-1} \ldots \circ a_1(x) - a_k \circ a_{k-1} \circ \ldots \circ a_1(x)\|_2 \\ &\leq \ell_k \|a_{k-1,\nu_{k-1}} \ldots \circ a_{1,\nu_1}(x) - a_{k-1} \ldots \circ a_1(x)\|_2 + \nu_k. \end{aligned}$$

Unrolling the recursion, we get
$$\|a_{\nu_{1:k}}(x) - a(x)\|_2 \leq \sum_{j=1}^k \nu_j \prod_{i=j+1}^k \ell_i. \tag{17}$$
Moreover we have by Lemma 19 that $a_{\nu_{1:k}}$ is L_a-smooth with
$$L_a = \sum_{j=1}^k \left(L_j + \frac{K_j}{\nu_j}\right) \left(\prod_{i=1}^{j-1} \ell_i\right)^2 \left(\prod_{i=j+1}^k \ell_i\right). \tag{18}$$
Denote $I = \{j \in \{1, \ldots , k\} : K_j > 0\}$. For $\mu > 0$, define for $j \in \{1 \ldots , k\}$,
$$\nu_j = \begin{cases} \mu/(|I| \prod_{i=j+1}^k \ell_i) & \text{if } j \in I \\ 0 & \text{if } j \notin I \end{cases},$$
and define $a_\mu = a_{\nu_{1:k}}$ as the smoothing of a. Then we have

1) a and a_ν are ℓ-Lipschitz continuous with $\ell = \prod_{j=1}^k \ell_j$ by Lemma 19,

2) a_ν approximates a up to an additive error μ by Eq. (17),
3) a_ν is $L + |I|K/\mu$-smooth with $L = \sum_{j=1}^k L_j \left(\prod_{i=1}^{j-1} \ell_i\right)^2 \left(\prod_{i=j+1}^k \ell_i\right), K = \ell^2 \sum_{j\in I} \frac{K_j}{\ell_j^2}$ by Eq. (18).

□

C. Chains of compositions

Proposition 7. *Consider a chain f of τ computations ϕ_t defined by*

$$\phi_t(x_{t-1}, u_t) = a_t(b_t(x_{t-1}, u_t)),$$

for $t \in \{1, \dots, \tau\}$, where $b_t \in \mathcal{B}_{L_{b_t}, l^u_{b_t}, l^x_{b_t}}$ and a_t is decomposed as

$$a_t = a_{t,k_t} \circ \ldots \circ a_{t,1},$$

with $a_{t,i} \in \mathcal{S}_{\ell_{a_{t,i}}, L_{a_{t,i}}, K_{a_{t,i}}}$. Consider $C = \bigotimes_{t=1}^\tau B_{R_t}(\mathbb{R}^{p_t})$. For any accuracy $\mu > 0$, Algo. 2 defines a smooth chain of computations f_μ, whose output approximates the output of f up to μ accuracy on C, i.e.

$$\forall x_0, u \in \mathbb{R}^{d_0} \times C, \qquad \|f_{\mu,\tau}(x_0, u) - f_\tau(x_0, u)\|_2 \le \mu.$$

The smoothness parameter of f_μ is of the form $L + K/\mu$, where L, K can be computed by Algo. 1 (see proof for an analytical estimate).

Proof. Fix $\mu_{t,j} > 0$ for $t \in \{1, \dots, \tau\}, j \in \{1, \dots, k_t\}$ and denote $a_{t,j,\mu_{t,j}}$ the smoothing of $a_{t,j}$. Denote $\tilde{f}$ the chain of computations defined for $u \in \mathbb{R}^p$ $x_0 \in \mathbb{R}^{d_0}$ by

$$\tilde{f}_t(x_0, u) = a_{t,\mu_t}(b_t(\tilde{f}_{t-1}(x_0, u), u_t)) \quad \text{for } t \in \{1, \dots, \tau\},$$

where $a_{t,\mu_t} = a_{t,k_t,\mu_{t,k_t}} \circ \ldots \circ a_{t,1,\mu_{t,1}}$ and $\tilde{f}_0(x_0, u) = x_0$. Define for simplicity in the following, $f_t = f_{t,x_0}$ and $\tilde{f}_t = \tilde{f}_{t,x_0}$. We have as in the proof of Lemma 25, for $t \in \{1, \dots, \tau\}$,

$$\begin{aligned}\|f_t(u) - \tilde{f}_t(u)\|_2 &\le \ell_{a_t} \|b_t(\tilde{f}_{t-1}(u), u_t) - b_t(f_{t-1}(u), u_t)\|_2 + \sum_{j=1}^{k_t} \mu_{t,j} \prod_{i=j+1}^{k_t} \ell_{a_{t,i}} \\ &\le \ell_{a_t}(R_t L_{b_t} + l^x_{b_t}) \|f_{t-1}(u) - \tilde{f}_{t-1}(u)\|_2 + \sum_{j=1}^{k_t} \mu_{t,j} \prod_{i=j+1}^{k_t} \ell_{a_{t,i}},\end{aligned}$$

where $\ell_{a_t} = \prod_{j=1}^{k_t} \ell_{a_{t,j}}$. Therefore we have

$$\|\tilde{f}_\tau(u) - f_\tau(u)\|_2 \le \sum_{t=1}^\tau \sum_{j=1}^{k_t} \mu_{t,j} \prod_{s=t+1}^\tau \ell_{a_s}(R_s L_{b_s} + l^x_{b_s}) \prod_{i=j+1}^{k_t} \ell_{a_{t,i}}.$$

For $\mu > 0$, taking

$$\mu_{t,j} = \begin{cases} \mu / (n \prod_{s=t+1}^\tau \ell_{a_s}(R_s L_{b_s} + l^x_{b_s}) \prod_{i=j+1}^{k_t} \ell_{a_{t,i}}) & \text{if } K_{t,j} > 0 \\ 0 & \text{if } K_{t,j} = 0, \end{cases}$$

where $n = \sum_{t=1}^\tau \sum_{j=1}^{k_t} \mathbf{1}_{K_{a_{t,j}} > 0}$ ensures then that

$$\|f_\tau(u) - \tilde{f}_\tau(u)\|_2 \le \mu.$$

Note that $\ell_{t,j}$ defined in Algo. 2 is equal to $\prod_{s=t+1}^\tau \ell_{a_s}(R_s L_{b_s} + l^x_{b_s}) \prod_{i=j+1}^{k_t} \ell_{a_{t,i}}$.

The smoothness parameters of the smoothing of the layers follow from Prop. 4. Namely define for $t \in \{1, \dots, \tau\}$,

$$\begin{aligned} m_t &= m_{t,k_t}, \\ m_{t,j} &= \min\{m_{a_{t,j}},\ \|a_{t,j}(0)\|_2 + \ell_{a_{t,j}} m_{t,j-1}\} \text{ for } j=1,\dots,k_t, \\ m_{t,0} &= L_{b_t} R_t m_{t-1} + l^u_{b_t} R_t + l^x_{b_t} m_{t-1} + \|\beta^0\|_2, \\ m_0 &= \|x\|_2, \\ \ell_t &= \left(\prod_{j=1}^{k_t} \ell_{a_{t,j}}\right) ((L_{b_t} R_t + l^x_{b_t}) \ell_{t-1} + L_{b_t} m_{t-1} + l^u_{b_t}), \\ \ell_0 &= 0. \end{aligned}$$

We then have that $\tilde{f}$ is $L + K/\mu$ smooth with $L = L_\tau$, $K = K_\tau$ where for $t \in \{1, \ldots, \tau\}$,

$$\begin{aligned}
L_t &= L_{t-1}(L_{b_t} R_t + l^x_{b_t}) \left(\prod_{j=1}^{k_t} \ell_{a_{t,j}}\right) \\
&\quad + (L_{b_t} R_t + l^x_{b_t})^2 L_{t,k_t} \ell_{t-1}^2 \\
&\quad + 2\left((L_{b_t} m_{t-1} + l^u_{b_t})(L_{b_t} R_t + l^x_{b_t}) L_{t,k_t} + L_{b_t}\left(\prod_{j=1}^{k_t} \ell_{a_{t,j}}\right)\right)\ell_{t-1} \\
&\quad + (L_{b_t} m_{t-1} + l^u_{b_t})^2 L_{t,k_t}, \\
L_{t,k_t} &= \sum_{j=1}^{k_t} L_{a_{t,j}} \left(\prod_{i=1}^{j-1} \ell_{t,i}\right)^2 \left(\prod_{i=j+1}^{k_t} \ell_{t,i}\right). \\
K_t &= K_{t-1}(L_{b_t} R_t + l^x_{b_t}) \left(\prod_{j=1}^{k_t} \ell_{a_{t,j}}\right) \\
&\quad + (L_{b_t} R_t + l^x_{b_t})^2 \ell_{t-1}^2 K_{t,k_t} \prod_{i=t+1}^{\tau} \ell_{a_i}(R_i L_{b_i} + l^x_{b_i}) \\
&\quad + 2\left((L_{b_t} m_{t-1} + l^u_{b_t})(L_{b_t} R_t + l^x_{b_t}) K_{t,k_t} \prod_{i=t+1}^{\tau} \ell_{a_i}(R_i L_{b_i} + l^x_{b_i})\right)\ell_{t-1} \\
&\quad + (L_{b_t} m_{t-1} + l^u_{b_t})^2 K_{t,k_t} \prod_{i=t+1}^{\tau} \ell_{a_i}(R_i L_{b_i} + l^x_{b_i}), \\
K_{t,k_t} &= |I_t| \left(\prod_{j=1}^{k_t} \ell_{a_{t,j}}\right)^2 \sum_{j \in I_t} \frac{K_{a_{t,j}}}{\ell^2_{a_{t,j}}} \\
I_t &= \{j \in \{1, \ldots, k_t\} : K_{a_{t,j}} > 0\}.
\end{aligned}$$

□

Lemma 8. *Consider the objective (5), assume the chain of computations f satisfies the assumptions of Prop. 7, and that $h^{(i)}$ is ℓ_h-Lipschitz continuous and L_h-smooth for any i. Consider $C = \bigotimes_{t=1}^{\tau} B_{R_t}(\mathbb{R}^{p_t})$. Then for any accuracy $\mu > 0$, the objective*

$$F_\mu(u) = \frac{1}{n}\sum_{i=1}^{n} h^{(i)}(f_{\nu,\tau}(x^{(i)}, u)),$$

with $\nu = \mu/\ell_h$, and $f_{\nu,\tau}$ computed by Algo. 2, is smooth on C and approximates F to μ accuracy on C, i.e., $|F(u) - F_\mu(u)| \le \mu$ for any $u \in C$.

Proof. The proof is immediate using the definition of $f_{\nu,\tau}$ and the Lipschitz-continuity of $h^{(i)}$. □

Lemma 9. *Consider the assumptions of Lemma 8 and let $\varepsilon > 0$. Assume that a first-order method applied to F_ε on C outputs a point $\hat{u}$ that is ε minimal on its neighborhood. Namely, there exists $\eta > 0$ such that*

$$F_\varepsilon(\hat{u}) - \min_{u \in B_\rho} F_\varepsilon(u) \le \varepsilon,$$

with $B_\rho = \{u \in \mathbb{R}^{\sum_{t=1}^{\tau} p_t} : \|\hat{u} - u\|_2 \le \eta\} \subset C$. Then the point $\hat{u}$ is also 3ε minimal for the original objective F on this neighborhood, i.e., we have

$$F(\hat{u}) - \min_{u \in B_\rho} F(u) \le 3\varepsilon.$$

Proof. Denoting $u^* = \arg\min_{u \in B_\rho} F(u)$, we have

$$F(\hat{u}) - F(u^*) \le F_\varepsilon(\hat{u}) - F_\varepsilon(u^*) + 2\varepsilon \le F_\varepsilon(\hat{u}) - \min_{u \in B_\rho} F_\varepsilon(u) + 2\varepsilon \le 3\varepsilon.$$

□