An Elementary Approach to Convergence Guarantees of Optimization Algorithms for Deep Networks

Vincent Roulet, Zaid Harchaoui
Department of Statistics, University of Washington, Seattle, USA

Abstract

We present an approach to obtain convergence guarantees of optimization algorithms for deep networks based on elementary arguments and computations. The convergence analysis revolves around the analytical and computational structures of optimization oracles central to the implementation of deep networks in machine learning software. We provide a systematic way to compute the smoothness constants that govern the convergence behavior of first-order optimization algorithms used to train deep networks. A diverse set of example components and architectures arising in modern deep networks intersperse the exposition to illustrate the approach.

1 Introduction

Deep networks have achieved remarkable performance in several application domains such as computer vision, natural language processing and genomics (Krizhevsky et al. 2012, Pennington et al. 2014, Duvenaud et al. 2015). A deep network can be framed as a chain of composition of modules, where each module is typically the composition of a non-linear activation function and an affine transformation. The last module in the chain is usually task-specific and can be expressed either in analytical form as in supervised classification or as the solution of an optimization problem in dimension reduction or clustering.

The optimization problem arising when training a deep network is often framed as a non-convex optimization problem, dismissing the structure of the objective yet central to the software implementation. Indeed optimization algorithms used to train deep networks proceed by making calls to first-order (or second-order) oracles relying on dynamic programming such as gradient back-propagation (Werbos 1994, Rumelhart et al. 1985, Lecun 1988). See also (Duda et al. 2012, Anthony & Bartlett 2009, Shalev-Shwartz & Ben-David 2014, Goodfellow et al. 2016) for an exposition and (Paszke et al. 2017, Abadi et al. 2015) for an implementation of gradient back-propagation for deep networks. We highlight here the elementary yet important fact that the chain-compositional structure of the objective naturally emerges through the smoothness constants governing the convergence guarantee of a gradient-based optimization algorithm. This provides a reference frame to relate the network topology and the convergence rate through the smoothness constants. This also brings to light the benefit of specific modules popular among practitioners to improve the convergence.

In Sec. 2, we define the parameterized input-output map implemented by a deep network as a chain-composition of modules and write the corresponding optimization objective consisting in learning the parameters of this map. In Sec. 3, we detail the implementation of first-order and second-order oracles by dynamic programming; the classical gradient back-propagation algorithm is recovered as a canonical example. Gauss-Newton steps can also be simply stated in terms of calls to a dynamic-programming-type procedure implemented in modern machine learning software libraries. In Sec. 4, we present the computation of the smoothness constants of several types of modules arising in deep networks and the resulting convergence guarantees for gradient descent. Finally, in Sec. 5, we present the application of the approach to derive the smoothness constants for the VGG architecture (Simonyan & Zisserman 2015). All proofs and notations are provided in the long version (Roulet & Harchaoui 2019).

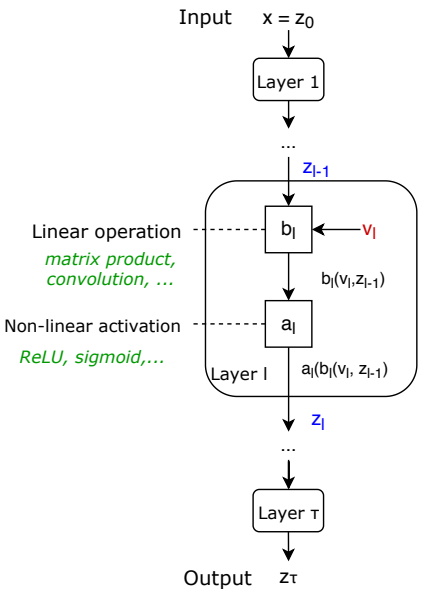

Figure 1: Deep network compositional structure.

2 Problem formulation

2.1 Deep network structure

A feed-forward deep network of depth τ can be described as a transformation of an input x into an output z_τ through the composition of τ blocks, called layers, illustrated in Fig. 1. Each layer is defined by a set of parameters v_l. For simple layers (see Sec. 2.3 for a formal and general definition), these parameters act on the input of the layer z_{l-1} through a linear operation b_l followed by a non-linear operation a_l, called an activation function. Formally, in that case, the l^{th} layer can be described as a function of its parameters v_l and a given input z_{l-1} that outputs z_l as

$$z_l = \phi_l(v_l, z_{l-1}) = a_l(b_l(v_l, z_{l-1})),$$

where b_l is a linear function of the inputs z_{l-1} and an affine function of the parameters v_l and a_l is a non-linear operation.

Learning a deep network consists in minimizing w.r.t. its parameters an objective involving n examples $x^{(1)}, \dots, x^{(n)} \in \mathbb{R}^d$. Formally, the problem is written

$$\begin{aligned} \min_{v_1,\dots,v_\tau} \quad & f(z_\tau^{(1)}, \dots, z_\tau^{(n)}) + r(v_1, \dots, v_\tau) \\ \text{subject to} \quad & z_l^{(i)} = \phi_l(v_l, z_{l-1}^{(i)}) \quad \text{for } l = 1, \dots, \tau \\ & z_0^{(i)} = x^{(i)}, \end{aligned} \tag{1}$$

where $v_l \in \mathbb{R}^{\rho_l}$ is the set of parameters at layer l whose dimension ρ_l can vary among layers, r is a regularization on the parameters of the network and the constraints in (1) are for any $i \in \{1, \dots, n\}$.

We are interested in the influence of the structure of the problem, i.e., the chain of compositions, on the optimization complexity of the problem. This leads to the definition of a chain of layers as follows.

Definition 2.1. *A function $\psi : \mathbb{R}^p \to \mathbb{R}^q$ is a* chain of τ layers*, with $k \leq p$, if it is defined by an input $x \in \mathbb{R}^{\delta_0}$ and τ functions $\phi_l : \mathbb{R}^{\rho_l} \times \mathbb{R}^{\delta_{l-1}} \to \mathbb{R}^{\delta_l}$ for $l = 1, \dots, \tau$ such that for $(v_1; \dots; v_\tau)$[1] $\in \mathbb{R}^p$ with $v_l \in \mathbb{R}^{\rho_l}$, the output of ψ is given by*

$$\begin{aligned} & \psi(w) = z_\tau, \\ \textit{with} \quad & z_l = \phi_l(v_l, z_{l-1}) \quad \textit{for } l = 1, \dots, \tau \\ & z_0 = x. \end{aligned} \tag{2}$$

By considering the concatenation of the parameters $w = (v_1; \dots; v_\tau)$ and the concatenation of the transformations of each input as a single transformation, i.e., $\psi(w) = (\psi^{(1)}(w), \dots, \psi^{(n)}(w))$ where $\psi^{(i)}$ is the chain of layers defined by the input $x^{(i)}$, the objective in (1) can be written as

$$\min_{w \in \mathbb{R}^p} \quad f(\psi(w)) + r(w), \tag{3}$$

where $\psi : \mathbb{R}^p \to \mathbb{R}^q$ is a chain of k layers[2], $r : \mathbb{R}^p \to \mathbb{R}$ is typically a decomposable differentiable function such as $r(w) = \sum_{l=1}^{\tau} \|v_l\|_2^2$ and we present examples of learning objectives below. Assumptions on differentiability and smoothness of the objective are detailed in Sec. 4.

[1] We denote by semi-columns the concatenation of variables.

[2] Note that if k is the number of classes, i.e., the output of each chain $\psi^{(i)}$, the output dimension of the chain ψ scales with the number of samples as $q = nk$.

2.2 Objectives

Supervised learning. For supervised learning, the objective can be decomposed as

$$f(\psi(w)) = \frac{1}{n}\sum_{i=1}^{n} f^{(i)}(\psi^{(i)}(w)), \tag{4}$$

where $f^{(i)}$ are losses on the labels predicted by the chain of layers, i.e., $f^{(i)}(\hat{y}^{(i)}) = \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$ where $y^{(i)}$ is the label of the input $x^{(i)}$ of the chain of layers $\psi^{(i)}$ with output $\hat{y}^{(i)}$ and $\mathcal{L}$ is a given loss.

Unsupervised learning. In unsupervised learning tasks the labels are unknown. The objective itself is defined through a minimization problem rather than through an explicit loss function. For example, a convex clustering objective is written

$$f(\psi(w)) = \min_{y_1,\dots,y_n\in\mathbb{R}^q} \sum_{i=1}^{n} \frac{1}{2}\|y^{(i)} - \psi^{(i)}(w)\|_2^2 + \sum_{i<j} \|y^{(i)} - y^{(j)}\|_2,$$

where $\psi^{(i)}(w)$ are n chains defined by inputs $x^{(i)}$, see (Hocking et al. 2011) for the linear formulation. We consider in (Roulet & Harchaoui 2019) different clustering objectives. Note that the classical ones (k-means, spectral clustering) are inherently non-smooth as they are defined as the minimization of a linear objective under constraints.

Auto-encoders. An auto-encoder seeks to reconstruct inputs $x^{(i)} \in \mathbb{R}^d$. The structure can be described as chain of layers $\psi^{(i)}$ (each fed with corresponding $x^{(i)}$) as defined in Def. 2.1 with the particularity that $\delta_0 = \delta_\tau = d$. The objective can then be written in the present framework as

$$f(\psi(w)) = \sum_{i=1}^{n} \|x^{(i)} - \psi^{(i)}(w)\|_2^2.$$

2.3 Layers

We describe in more detail deep network layers. In general, each layer is described by (i) a normalization of the input, (ii) a linear operation, (iii) a non-linear operation, (iv) a pooling operation, i.e., a layer can be written as

$$\phi_l(v_l, z_{l-1}) = \pi_l\Big(a_l\big(b_l(v_l, c_l(z_{l-1}))\big)\Big) \tag{5}$$

where $c_l : \mathbb{R}^{\delta_{l-1}} \to \mathbb{R}^{\delta_{l-1}}$ is a *normalization* operation,

$$b_l(v_l, z_{l-1}) = b_l^1(v_l, z_{l-1}) + b_l^0(v_l) \tag{6}$$

with $b_l^1 : \mathbb{R}^{\rho_l} \times \mathbb{R}^{\delta_{l-1}} \to \mathbb{R}^{\eta_l}$ bilinear, $b_l^0 : \mathbb{R}^{\rho_l} \to \mathbb{R}^{\eta_l}$ linear, $a_l : \mathbb{R}^{\eta_l} \to \mathbb{R}^{\eta_l}$ is an *activation* function and $\pi_l : \mathbb{R}^{\eta_l} \to \mathbb{R}^{\delta_l}$ is a *pooling* operation. We present common examples of such functions, a list is detailed in (Roulet & Harchaoui 2019) with the smoothness properties of each function.

Linear operations. A *fully connected* layer taking a batch of m inputs of dimension d is written

$$\tilde{Z} = V^\top Z + v^0 \mathbf{1}_m^\top \tag{7}$$

where $Z \in \mathbb{R}^{d\times m}$ is the batch of inputs and $V \in \mathbb{R}^{d\times\tilde{d}}$, $v^0 \in \mathbb{R}^{\tilde{d}}$ define an affine transformation. We dropped the dependency w.r.t. the layer l and denoted by $\tilde{\cdot}$ the quantities characterizing the output. In the vocabulary of Def. 2.1, we have

- $z_{l-1}=\mathrm{Vec}(Z), \delta_{l-1}=md, z_l=\mathrm{Vec}(\tilde{Z}), \delta_l=m\tilde{d}$,
- $v_l = \mathrm{Vec}((V^\top, v^0)^\top), \rho_l = \tilde{d}(d+1)$,
- $b_l^1(v_l, z_{l-1}) = \mathrm{Vec}(V^\top Z), b_l^0(v_l) = \mathrm{Vec}(v^0 \mathbf{1}_m^\top)$

Note that the dimension of the parameters scales roughly as the square of the dimension of the inputs. A matrix product can easily be translated into a bilinear operation on the vectorized matrices, see (Roulet & Harchaoui 2019).

A *convolutional* layer convolves a batch of m inputs (images or signals) of dimension d stacked as $Z = (z_1, \dots, z_m) \in \mathbb{R}^{d\times m}$ with n^f affine filters of size s^f defined by weights $V = (v_1, \dots, v_{n^f}) \in \mathbb{R}^{s^f\times n^f}$ and offset $v^0 = (v_1^0, \dots, v_{n^f}^0) \in \mathbb{R}^{n^f}$ through n^p patches. The k^{th} output of the convolution of the i^{th} input by the j^{th} filter reads

$$\Xi_{i,j,k} = v_j^\top \Pi_k z_i + v_j^0 \tag{8}$$

where $\Pi_k \in \mathbb{R}^{s^f\times d_{l-1}}$ extracts a patch of size s^f at a given position. The output $\tilde{Z} = (\tilde{z}_1, \dots, \tilde{z}_m)$

is then given by the concatenation of each input, i.e., $\tilde{z}_{i,k+n^p(j-1))} = \Xi_{i,j,k}$. In the following we denote by

$$V \star_{n^p} Z$$

any convolution of m inputs by n^f filters through n^p number of patches (the convolution itself may depend on stride and padding options characterized by the number of patches n^p, yet only this number will affect the complexities). The corresponding linear operations are obtained using $z_{l-1} = \mathrm{Vec}(Z) \in \mathbb{R}^{md}$, $z_l = \mathrm{Vec}(\tilde{Z}) \in \mathbb{R}^{m\tilde{d}}$ where $\tilde{d} = n^p n^f$ and $v_l = \mathrm{Vec}((V^\top, v^0)^\top) \in \mathbb{R}^{(s^f+1)n^f}$.

Activation functions. We consider element-wise activation functions $a : \mathbb{R}^\delta \to \mathbb{R}^\delta$ such that for a given $z = (z_1, \dots, z_\delta) \in \mathbb{R}^\delta$,

$$a(z) = (\alpha(z_1), \dots, \alpha(z_\delta)) \tag{9}$$

for a given scalar function α such as $\alpha(x) = \max(x, 0)$ for the Rectified Linear Unit (ReLU) or the sigmoid function $\alpha(x) = (1 + \exp(-x))^{-1}$. To account for non-differentiable layers, we consider the smoothing counterpart using inf-convolution, see (Roulet & Harchaoui 2019).

Normalization functions. We consider either the case where the inputs are not normalized, i.e., $c(z) = z$ or the case where the inputs are normalized through batch-normalization, i.e., batch of input $Z \in \mathbb{R}^{d\times m}$ outputs $\tilde{Z}$ defined by

$$(\tilde{Z})_{ij} = \frac{Z_{ij} - \mu_i}{\sqrt{\epsilon + \sigma_i^2}}, \tag{10}$$

$$\text{where} \qquad \mu_i = \frac{1}{m}\sum_{j=1}^m Z_{ij}, \quad \sigma_i^2 = \frac{1}{m}\sum_{j=1}^m (Z_{ij} - \mu_i)^2,$$

with $\epsilon > 0$, such that $c(z) = \mathrm{Vec}(\tilde{Z})$ for $z = \mathrm{Vec}(Z)$.

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 a batch of inputs $Z \in \mathbb{R}^{d\times m}$, the average pooling with a patch size s^f for inputs with n^f channels through n^p patches outputs

$$\tilde{Z} = (\mathbf{1}_{s^f}\, \mathbf{1}_{n^f}^\top / s^f) \star_{n^p} Z \tag{11}$$

where the convolution $\star_{n^p}$ induces a reduction of dimension, i.e., the number of patches is $n^p \leq d/n^f$.

3 Oracle arithmetic complexity

For each class of optimization algorithm considered (gradient descent, Gauss-Newton, Newton), we define the appropriate optimization oracle called at each step of the optimization algorithm which can be efficiently computed through a dynamic programming procedure akin to the classical automatic differentiation procedure. All optimization oracles can be formally defined as the minimization of the objective with an additional proximal term.

Oracle reformulations. For a function f, we denote

$$\ell_f(y; x) = f(x) + \nabla f(x)^\top (y - x)$$
$$q_f(y; x) = f(x) + \nabla f(x)^\top (y - x) + \frac{1}{2}(y - x)^\top \nabla^2 f(x)(y - x)$$

the linear and quadratic approximations respectively of f around x. On a point w_t, given a step-size γ,

(i) a *gradient* step is defined as

$$w_{t+1} = \arg\min_{w\in\mathbb{R}^p} \ell_{f\circ\psi}(w; w_t) + \ell_r(w; w_t) + \frac{1}{2\gamma}\|w - w_t\|_2^2, \tag{12}$$

(ii) a (regularized) *Gauss-Newton* is defined as

$$w_{t+1} = \arg\min_{w\in\mathbb{R}^p} q_f(\ell_\psi(w; w_t); \psi(w_t)) + q_r(w; w_t) + \frac{1}{2\gamma}\|w - w_t\|_2^2, \tag{13}$$

(iii) a *Newton* step is defined as

$$w_{t+1} = \arg\min_{w\in\mathbb{R}^p} q_{f\circ\psi}(w; w_t) + q_r(w; w_t) + \frac{1}{2\gamma}\|w - w_t\|_2^2. \tag{14}$$

All those steps amount to solving quadratic problems on a linearized network as shown in the following proposition.

Proposition 3.1. *Let $w_t = (v_1; \dots; v_\tau)$ and $z_0, \dots, z_\tau$ be defined by the chain of compositions in (2) applied to w_t.*

Assume r to be decomposable as $r(w_t) = \sum_{l=1}^{\tau} r_l(v_l)$. Gradient (12), Gauss-Newton (13) and Newton (14) steps are given as $w_{t+1} = w_t + \tilde{w}^*$ where $\tilde{w}^* = (\tilde{v}_1^*; \dots; \tilde{v}_\tau^*)$ is the solution of

$$\min_{\substack{\tilde{v}_1,\dots,\tilde{v}_\tau \\ \tilde{z}_0,\dots,\tilde{z}_\tau}} \quad \sum_{l=1}^{\tau} \frac{1}{2}\tilde{z}_l^\top P_l \tilde{z}_l + p_l^\top \tilde{z}_l + \tilde{z}_{l-1}^\top R_l \tilde{v}_l + \frac{1}{2}\tilde{v}_l^\top Q_l \tilde{v}_l + q_l^\top \tilde{v}_l + \frac{1}{2\gamma}\|\tilde{v}_l\|_2^2 \qquad (15)$$

$$\text{subject to} \quad \tilde{z}_l = A_l \tilde{z}_{l-1} + B_l \tilde{v}_l \quad \text{for} \quad l \in \{1,\dots,\tau\}$$
$$\tilde{z}_0 = 0,$$

where $\tilde{v}_l \in \mathbb{R}^{\rho_l}, \tilde{z}_l \in \mathbb{R}^{\delta_l}$ with ρ_l, δ_l defined by ψ in Def. 2.1,

$$A_l = \nabla_{z_{l-1}} \phi_l(v_l, z_{l-1})^\top, \quad B_l = \nabla_{v_l} \phi_l(v_l, z_{l-1})^\top$$
$$p_\tau = \nabla f(\psi(w_t)), \quad p_l = 0 \quad \text{for } l \neq \tau$$
$$q_l = \nabla r_l(v_l)$$

1. *for gradient steps (12), $P_l = 0$, $R_l = 0$, $Q_l = 0$,*
2. *for Gauss-Newton steps (13), $P_\tau = \nabla^2 f(\psi(w_t))$, $P_l = 0$ for $l \neq \tau$, $R_l = 0$, $Q_l = \nabla^2 r_l(v_l)$,*
3. *for Newton steps (14), defining λ_l as $\lambda_\tau = \nabla f(\psi(w_t))$ and $\lambda_{l-1} = \nabla_{z_{l-1}} \phi_l(v_l, z_{l-1})\lambda_l$ for $l \in \{1,\dots,\tau\}$, $P_\tau = \nabla^2 f(\psi(w_t))$, $P_{l-1} = \nabla^2_{z_{l-1} z_{l-1}} \phi_l(v_l, z_{l-1})[\cdot,\cdot,\lambda_l]$ for $l \in \{1,\dots,\tau\}$, $R_l = \nabla^2_{z_{l-1} v_l} \phi_l(v_l, z_{l-1})[\cdot,\cdot,\lambda_l]$, $Q_l = \nabla^2 r_l(v_l) + \nabla^2_{v_l v_l} \phi_l(v_l, z_{l-1})[\cdot,\cdot,\lambda_l]$.*

Quadratic problems with linear compositions can be solved efficiently by dynamic programming such that the complexity of all these steps is linear w.r.t. to the length τ of the chain. We present in (Roulet & Harchaoui 2019) the detailed computation of a Newton step. This involves the inversion of intermediate quadratic costs at each layer. Gauss-Newton steps can also be solved by dynamic programming and can be more efficiently implemented using an automatic differentiation procedure as we explain below.

Forward-backward algorithm. For gradient steps the resolution of Eq. (15) by dynamic programming amounts to the classical forward-backward algorithm detailed below as shown in (Roulet & Harchaoui 2019). Given a chain of τ layers $\psi : \mathbb{R}^p \to \mathbb{R}^q$ as defined in Def. 2.1 and a differentiable objective f, the forward-backward algorithm computes the gradient $\nabla[f \circ \psi](w) = (g_1; \dots; g_\tau) \in \mathbb{R}^p$ with $g_l = \nabla_{v_l}[f \circ \psi](w) \in \mathbb{R}^{\rho_l}$ as follows.

(A) For $l = 1, \dots, \tau$, starting with $z_0 = x$, compute $z_l = \phi_l(v_l, z_{l-1})$ and store $\nabla_{v_l} \phi_l(v_l, z_{l-1})$, $\nabla_{z_{l-1}} \phi_l(v_l, z_{l-1})$.

(B) For $l = \tau, \dots, 1$, starting from $\lambda_\tau = \nabla f(z_\tau)$, compute $\lambda_{l-1} = \nabla_{z_{l-1}} \phi_l(v_l, z_{l-1})\lambda_l$ and output $g_l = \nabla_{v_l} \phi_l(v_l, z_{l-1})\lambda_l$.

Without additional information on the structure of the layers, the cost of the backward pass (B) is of the order of

$$\mathcal{O}\left(\sum_{l=1}^{\tau} \delta_{l-1}\delta_l + \rho_l \delta_l\right)$$

elementary operations during the backward pass. Compared to a naive implementation of the chain-rule that would compute $\nabla\psi(w)$ and $\nabla f(w)$ separately, then multiply both at a cost $\mathcal{O}((\sum_{l=1}^{\tau} \delta_l)(\sum_{l=1}^{\tau} \rho_l))$, the forward backward algorithm exploits well the composite structure of the chain.

For chain of layers of the form (5), this cost can be refined as shown in (Roulet & Harchaoui 2019). Specifically, for a chain of fully-connected layers with element-wise activation function, no normalization or pooling, the cost of the backward pass is then of the order of $\mathcal{O}\left(\sum_{l=1}^{\tau}(2d_{l-1}+1)(md_l)^2\right)$ elementary operations. For a chain of convolutional layers with element-wise activation function, no normalization or pooling, the cost of the backward pass is of the order of $\mathcal{O}\left(\sum_{l=1}^{\tau}(2n_l^p n_l^f s_l^f + n_l^p n_l^f)m^2 d\right)$ elementary operations.

Gauss-Newton by automatic-differentiation. The Gauss-Newton step can also be solved by making calls to an automatic differentiation procedure as shown in (Roulet et al. 2019) and stated in the framework considered in this paper. We present the definition of an automatic differentiation oracle below.

Definition 3.2 ((Roulet et al. 2019, Definition 3.3))**.** *An* automatic differentiation oracle *is a procedure that, given a differentiable chain of compositions $\psi : \mathbb{R}^p \to \mathbb{R}^q$ as defined in Def. 2.1, and $w \in \mathbb{R}^p$ computes*

$$s \to \nabla\psi(w)s \qquad \text{for any } s \in \mathbb{R}^q.$$

Proposition 3.3. *Consider the Gauss-Newton-step (13) for convex objective f and convex decomposable regularization $r(w) = \sum_{l=1}^{\tau} r_l(v_l)$ for $w = (v_1; \dots; v_\tau)$. We have that*

1. *the Gauss-Newton-step amounts to solving*
$$\min_{s \in \mathbb{R}^q} \tilde{q}_f^\star(s) + \tilde{q}_r^\star(-\nabla\psi(w_t)s) \tag{16}$$
where $\tilde{q}_f(y) = q_f(\psi(w_t) + y; \psi(w_t))$, $\tilde{q}_r(w) = q_r(w_t + w; w_t) + \|w\|_2^2/2$ and for a function f we denote by $f^\star$ its convex conjugate,
2. *the Gauss-Newton-step reads $w_{t+1} = w_t + \nabla\tilde{q}_r^\star(-\nabla\psi(w_t)s^*)$ where s^* is the solution of (16),*
3. *the dual problem (16) can be solved by $2q + 1$ calls to an automatic differentiation procedure.*

Proposition 3.3 shows that a Gauss-Newton step is only $2q + 1$ times more expansive than a gradient-step. Precisely, for a deep network with a supervised objective, we have $q = nk$ where n is the number of samples and k is the number of classes. A gradient step makes then one call to an automatic differentiation procedure to get the gradient of the batch and the Gauss-Newton method will then make $2nk + 1$ more calls. If mini-batch Gauss-Newton steps are considered then the cost reduces to $2mk+1$ calls to an automatic differentiation oracle, where m is the size of the mini-batch of the examples.

4 Optimization complexity

The convergence guarantee of a first-order method towards a ε-stationary point is governed by the smoothness property of the objective, i.e., the Lipschitz continuity of the function itself or its gradient when defined. In the following, for a function f, we denote by

$$\begin{aligned} m_f &= \sup_{x \in \operatorname{dom} f} |f(x)| \\ \ell_f &= \sup_{x,y \in \operatorname{dom} f,\, x \neq y} \frac{|f(x) - f(y)|}{\|x - y\|_2} \\ L_f &= \sup_{x,y \in \operatorname{dom} f,\, x \neq y} \frac{\|\nabla f(x) - \nabla f(y)\|_2}{\|x - y\|_2} \end{aligned} \tag{17}$$

a bound on the function, its Lipschitz-continuity parameter and the Lipschitz-continuity parameter of its gradient (if it is defined) respectively. We study smoothness properties of the functions w.r.t. the Euclidean norm of their variables. In particular, for deep networks it means that we consider the Euclidean norm of the vectorized batch of inputs or images. For a class of functions $\mathcal{C}$ we denote by subscripts the properties that the functions possess. For example, we denote by $\mathcal{C}_{\ell,L}$ the set of differentiable functions that are ℓ-Lipschitz continuous and L-smooth w.r.t. the Euclidean norm, i.e., with L-Lipschitz continuous gradients as defined in Eq. (17).

Note that the propositions below give *upper bounds* on the smoothness of the function achieved through chain-composition. For a trivial composition such as $f \circ f^{-1}$, the upper bound is clearly loose. The upper bounds we present here are informative for non-trivial architectures.

Smoothness of chain of layers. We first present a general result for chain of layers without specific structure. Although the result is not readily applicable to deep networks, it clarifies the recurrence used to compute the smoothness of a chain of compositions.

Proposition 4.1. *Consider a chain ψ of τ layers as defined in Def. 2.1, by layers $\phi_l \in \mathcal{C}_{\ell_{\phi_l}, L_{\phi_l}}$.*

(i) An upper-bound on the Lipschitz-continuity of the chain ψ is given by $\ell_\psi = \ell_\tau$, where for $l \in \{1, \dots, \tau\}$,
$$\begin{aligned} \ell_l &= \ell_{\phi_l} + \ell_{l-1}\ell_{\phi_l} \\ \ell_0 &= 0. \end{aligned}$$

(ii) An upper-bound on the smoothness of the chain ψ is given by $L_\psi = L_\tau$, where for $l \in \{1, \dots, \tau\}$,
$$\begin{aligned} L_l &= L_{l-1}\ell_{\phi_l} + L_{\phi_l}(1 + \ell_{l-1})^2 \\ L_0 &= 0. \end{aligned}$$

Layers of deep networks are a priori not Lipschitz continuous. To get an estimate of their properties we need to dive into their specific parametrization and to consider their properties on compact sets as tackled in the following proposition. We denote by $\mathcal{B}_{\ell,L}$ the set of functions b defined as in (6) by a L-smooth bilinear function b^1 and a ℓ-Lipschitz continuous linear function b^0.

Proposition 4.2. *Consider a chain ψ of τ layers as defined in Def. 2.1 whose layers ϕ_l are defined as in (5) by $c_l \in \mathcal{C}_{m_{c_l}, \ell_{c_l} L_{c_l}}$, $b_l \in \mathcal{B}_{\ell_{b_l}, L_{b_l}}$, $a_l \in \mathcal{C}_{\ell_{a_l}, L_{a_l}}$ and $\pi_l \in \mathcal{C}_{\ell_{\pi_l}, L_{\pi_l}}$. Denote $C = \{w = (v_1; \dots; v_\tau) : \|v_l\|_2 \leq \mu_l\}$.*

(i) An upper-bound on the output of the chain ψ on C is

given by $m_\psi(C) = m_\tau$ *where for* $l \in \{1, \dots, \tau\}$,

$$m_l = \ell_{a_l} \ell_{\pi_l} \mu_l (\tilde{m}_{l-1} L_{b_l} + \ell_{b_l}) + \tilde{a}_{l,0}$$
$$m_0 = \|x\|_2,$$

where $\tilde{m}_{l-1} = \min\{(\ell_{c_l} m_{l-1} + c_{l,0}), m_{c_l}\}$, $\tilde{a}_{l,0} = \|\pi_l \circ a_l(0)\|_2$, $c_{l,0} = \|c_l(0)\|_2$.

(ii) An upper-bound on the Lipschitz-continuity of the chain ψ *on* C *is given by* $\ell_\psi(C) = \ell_\tau$, *where for* $l \in \{1, \dots, \tau\}$,

$$\ell_l = L_{b_l} \ell_{\pi_l} \ell_{a_l} (\ell_{l-1} \ell_{c_l} \mu_l + \tilde{m}_{l-1}) + \ell_{b_l} \ell_{\pi_l} \ell_{a_l} \quad (18)$$
$$\ell_0 = 0.$$

(iii) An upper-bound on the smoothness of the chain ψ *on* C *is given by* $L_\psi(C) = L_\tau$, *where for* $l \in \{1, \dots, \tau\}$,

$$\begin{aligned} L_l = {} & L_{l-1} L_{b_l} \mu_l \ell_{\tilde{a}_l} \ell_{c_l} \\ & + (L_{\tilde{a}_l} (\ell_{c_l} L_{b_l} \mu_l)^2 + L_{c_l} L_{b_l} \mu_l \ell_{\tilde{a}_l}) \ell_{l-1}^2 \\ & + 2(L_{\tilde{a}_l} \ell_{c_l} L_{b_l} \mu_l (L_{b_l} \tilde{m}_{l-1} + \ell_{b_l}) + L_{b_l} \ell_{c_l} \ell_{\tilde{a}_l}) \ell_{l-1} \\ & + L_{\tilde{a}_l} (L_{b_l} \tilde{m}_{l-1} + \ell_{b_l})^2, \end{aligned} \quad (19)$$
$$L_0 = 0,$$

where $\ell_{\tilde{a}_l} = \ell_{a_l} \ell_{\pi_l}$ *and* $L_{\tilde{a}_l} = L_{\pi_l} \ell_{a_l}^2 + L_{a_l} \ell_{\pi_l}$.

Convergence rate to a stationary point. We recall the stationary convergence result of a gradient descent given the smoothness property. We denote $\mathcal{C}_{\ell,L}(C)$ the set of functions ℓ Lipschitz continuous and L smooth on a compact set C.

Proposition 4.3. *Consider a composite problem*

$$\min_{w \in \mathbb{R}^p} \quad f(\psi(w)) + r(w),$$
$$\textit{subject to} \quad w \in C$$

where $\psi \in \mathcal{C}_{\ell_\psi, L_\psi}(C)$, $f \in \mathcal{C}_{\ell_f, L_f}(\psi(C))$, $r \in \mathcal{C}_{L_r}(C)$ *and* $C = \{w = (v_1; \dots; v_\tau) : \|v_l\|_2 \leq \mu_l\}$, *a projected gradient descent with step-size* $\gamma = (L_\psi \ell_f + \ell_\psi^2 L_f + L_r)^{-1}$ *converges to an* ε*-stationary point in at most*

$$\mathcal{O}\left(\frac{L_\psi \ell_f + \ell_\psi^2 L_f + L_r}{\epsilon^2}\right)$$

iterations.

Note that, since gradient descent guarantees monotonic decrease, one can replace C in the above proposition by the initial sub-level set of the objective $f \circ \psi + r$. A backtracking-line search can be incorporated into the gradient descent leading to the same rate. A similar result can be obtained for stochastic optimization.

5 Application

We apply our framework to assess the smoothness properties of the Visual Geometry Group (VGG) deep network used for image classification (Simonyan & Zisserman 2015).

5.1 VGG network

The VGG Network is a 16-layer convolutional network for image classification that achieved state-of-the-art performance at the ImageNet competition at the time. Its architecture is detailed in (Roulet & Harchaoui 2019). We consider in the following smoothness properties for mini-batches with size $m = 128$, i.e., by concatenating m chains of layers $\psi^{(i)}$ each defined by a different input[3]. These define incremental gradient oracles and their smoothness control the optimization process analogously as in Prop. 4.3.

Smoothness computations. To compute the Lipschitz-continuity and smoothness parameters, we recall the list of Lipschitz continuity and smoothness constants of each layer of interest. For the bilinear and linear operations we denote by L the smoothness of the bilinear operation and by ℓ the Lipschitz-continuity of the linear operation. We denote by n^p number of patches of the convolution operation. The smoothness constants of interest are

1. $\ell_{\text{conv}} = \sqrt{mn^p}$, $L_{\text{conv}} = \sqrt{n^p}$,
2. $\ell_{\text{full}} = \sqrt{m}$, $L_{\text{full}} = 1$,
3. $\ell_{\text{ReLu}} = 1$, L_{ReLu} not defined,
4. $\ell_{\text{softmax}} = 2$, $L_{\text{softmax}} = 4$,
5. $\ell_{\text{maxpool}} = 1$, L_{maxpool} not defined,
6. $\ell_{\log} = 2$, $L_{\log} = 2$.

A Lipschitz-continuity estimate of this architecture can then be computed using Prop. 4.2 on a Cartesian product of balls $C = \{w = (v_1; \dots; v_{16}) : \|v_l\|_2 \leq \mu\}$ for $\mu = 1$ for example.

[3] This highlights the impact of the size of the mini-batch for batch-normalization.

5.2 Variations of VGG

Smooth VGG. First, the VGG architecture can be made continuously differentiable by considering the soft-plus activation instead of the ReLU activation and average pooling instead of the max-pooling operation. As shown in (Roulet & Harchaoui 2019), we have

1. $\ell_{\text{avgpool}} = 1$, $L_{\text{avgpool}} = 0$,
2. $\ell_{\text{softplus}} = 2$, $L_{\text{softplus}} = 4$.

Denoting ℓ_{VGG} and $\ell_{\text{VGG-smooth}}$ the Lipschitz-continuity estimates of the original VGG network and the modified original network on a Cartesian product of balls $C = \{w = (v_1; \dots; v_{16}) : \|v_l\|_2 \leq 1\}$, we get using Prop. 4.2,

$$\ell_{\text{VGG}} \approx \ell_{\text{VGG-smooth}}$$

The only difference between the two computations resides in the constant $a_{\text{softplus}}(0) \neq 0$.

Batch-normalization effect. We can also compare the smoothness properties of the smoothed network with the same network modified by adding the batch-normalization layer for m inputs and ϵ normalization parameter at each convolutional layer. As shown in (Roulet & Harchaoui 2019) , the batch-normalization satisfies

1. $m_{\text{batch}}{=}dm$, $\ell_{\text{batch}}{=}2\epsilon^{-1/2}$, $L_{\text{batch}}{=}2m^{-1/2}\epsilon^{-1}$.

Denoting $\ell_{\text{VGG-smooth}}$, $L_{\text{VGG-smooth}}$ and $\ell_{\text{VGG-batch}}$, $L_{\text{VGG-batch}}$ the Lipschitz-continuity and smoothness estimates of the smoothed VGG network with and without batch-normalization respectively on a Cartesian product of balls $C = \{w = (v_1; \dots; v_{16}) : \|v_l\|_2 \leq 1\}$, we get using Prop. 4.2,

$$\begin{aligned} \text{for} \quad \epsilon = 10^{-2}, \quad & \ell_{\text{VGG-smooth}} \leq \ell_{\text{VGG-batch}} \\ & L_{\text{VGG-smooth}} \leq L_{\text{VGG-batch}} \\ \text{for} \quad \epsilon = 10^{2}, \quad & \ell_{\text{VGG-smooth}} \geq \ell_{\text{VGG-batch}} \\ & L_{\text{VGG-smooth}} \geq L_{\text{VGG-batch}} \end{aligned}$$

Intuitively, the batch-norm bounds the output of each layer, mitigating the increase of $\tilde{m}_l$ in Eq. (18) and (19). Yet, for a small ϵ, this effect is balanced by the non-smoothness of the batch-norm layer (which for $\epsilon \to 0$ tends to have an infinite slope around 0).

Acknowledgments. This work was supported by NSF CCF-1740551, NSF DMS-1839371, the program "Learning in Machines and Brains", and faculty research awards.

References

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

Anthony, M. & Bartlett, P. (2009), *Neural Network Learning: Theoretical Foundations*, Cambridge University Press.

Duda, R., Hart, P. & Stork, D. (2012), *Pattern classification*, 2nd edn, John Wiley & Sons.

Duvenaud, D. K., Maclaurin, D., Iparraguirre, J., Bombarell, R., Hirzel, T., Aspuru-Guzik, A. & Adams, R. P. (2015), Convolutional networks on graphs for learning molecular fingerprints, *in* 'Advances in Neural Information Processing Systems 28'.

Goodfellow, I., Bengio, Y. & Courville, A. (2016), *Deep Learning*, The MIT Press.

Hocking, T. D., Joulin, A., Bach, F. & Vert, J.-P. (2011), Clusterpath: an algorithm for clustering using convex fusion penalties, *in* 'Proceedings of the 28th International Conference on International Conference on Machine Learning'.

Krizhevsky, A., Sutskever, I. & Hinton, G. E. (2012), ImageNet classification with deep convolutional neural networks, *in* 'Advances in Neural Information Processing Systems 25'.

Lecun, Y. (1988), A theoretical framework for back-propagation, *in* '1988 Connectionist Models Summer School, CMU, Pittsburg, PA'.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L. & Lerer, A. (2017), 'Automatic differentiation in PyTorch'.

Pennington, J., Socher, R. & Manning, C. D. (2014), GloVe: Global vectors for word representation, *in* 'Empirical Methods in Natural Language Processing (EMNLP)', pp. 1532–1543.

Roulet, V. & Harchaoui, Z. (2019), 'An elementary approach to convergence guarantees of optimization algorithms for deep networks', *arXiv preprint* .

Roulet, V., Srinivasa, S., Drusvyatskiy, D. & Harchaoui, Z. (2019), Iterative linearized control: stable algorithms and complexity guarantees, *in* 'Proceedings of the 36th International Conference on Machine Learning'. Long version.

Rumelhart, D. E., Hinton, G. E. & Williams, R. J. (1985), Learning internal representations by error propagation, Technical report, California Univ San Diego La Jolla Inst for Cognitive Science.

Shalev-Shwartz, S. & Ben-David, S. (2014), *Understanding Machine Learning: From Theory to Algorithms*, Cambridge University Press.

Simonyan, K. & Zisserman, A. (2015), Very deep convolutional networks for large-scale image recognition, *in* 'International Conference on Learning Representations'.

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