An Inertial Newton Algorithm for Deep Learning

Camille Castera
IRIT, Université de Toulouse,
CNRS, Toulouse, France
camille.castera@irit.fr

Jérôme Bolte[*]
Toulouse School of Economics
Université Toulouse 1 Capitole, France
jerome.bolte@ut-capitole.fr

Cédric Févotte[*]
IRIT, Université de Toulouse,
CNRS, Toulouse, France
cedric.fevotte@irit.fr

Edouard Pauwels[*]
IRIT, Université de Toulouse, CNRS,
DEEL, IRT Saint Exupery, Toulouse, France
edouard.pauwels@irit.fr

Abstract

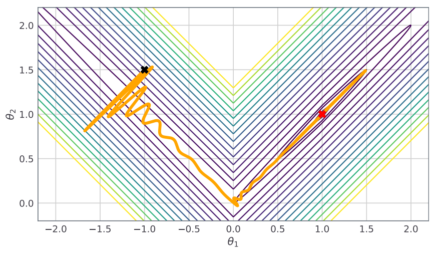
We introduce an inertial second-order method for machine learning, exploiting the geometry of the loss function while requiring only stochastic approximations function values and generalized gradients. The method features a simple mechanical interpretation and we describe promising numerical results on deep learning benchmarks. We give convergence guarantees in a theoretical framework encompassing most deep learning losses under very mild assumptions.

1 Introduction

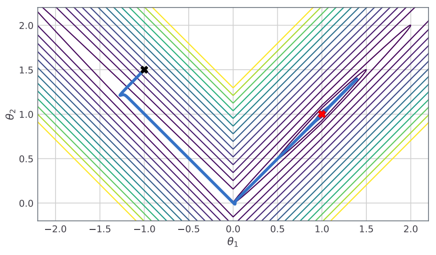
Training a deep neural network (DNN) is a challenging task that involves minimizing nonsmooth nonconvex functions which may have millions of parameters. Moreover, due to large datasets, stochastic approximations (mini-batch approaches) are necessary to compute the loss function or its gradient. In this context, computing the second-order derivative/objects (as the Hessian) of the loss function is inconceivable and encourages the design of algorithms that only require approximations of the gradient. To circumvent this problem several algorithms have been proposed, the most popular are probably ADAGRAD [13] and ADAM [14]. The first method uses geometrical information to calibrate the stepsizes while the second one exploits inertial ideas à la "'Heavy Ball with Friction" [18]. We propose a new algorithm which takes advantage of both inertial and second-order features while using only (sub)gradient approximations. We show promising preliminary experiments and provide cutting edge convergence guarantees for general nonsmooth nonconvex learning problems.[2]

Setting. In the rest of the paper, we consider DNNs which are locally Lipschitz continuous in their parameters $f : (x, \theta) \in \mathbb{R}^M \times \mathbb{R}^P \mapsto y \in \mathbb{R}^D$. The variable $\theta \in \mathbb{R}^P$ represents parameters of the model (P can be very large), while $x \in \mathbb{R}^M$ and $y \in \mathbb{R}^D$ represent input and output data. For instance, x may represent an image and y a label explaining some of its content. This encompasses many types of networks (e.g., a composition of feed-forward, convolutional, recurrent networks with ReLU, sigmoid, or tanh activation functions). Considering a dataset of N samples $(x_n, y_n)_{n=1,\dots,N}$. Training the NN amounts to finding a value of the parameter θ such that, for each input data x_n of the dataset, the output $f(x_n, \theta)$ of the model predicts the value y_n with good accuracy. To do so, we follow the traditional approach of minimizing an empirical risk loss function:
$\mathbb{R}^P \ni \theta \mapsto \mathcal{J}(\theta) = \sum_{n=1}^{N} l(f(x_n, \theta), y_n)$, where $l : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}$ is a locally Lipschitz continuous dissimilarity measure (e.g., distances on $\mathbb{R}^D$: $l(y_1, y_2)^2 = \|y_1 - y_2\|_2^2$).

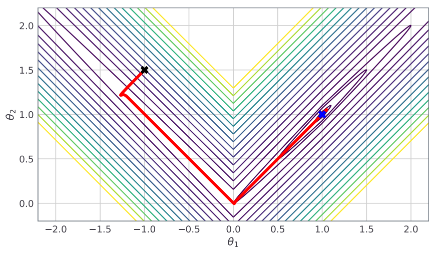
[*]Last three authors are listed in alphabetical order.
[2]This work is adapted from an extended version [8].



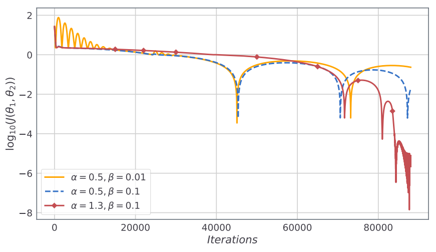
(a) $\alpha = 0.5,\ \beta = 0.01$



(b) $\alpha = 0.5,\ \beta = 0.1$



(c) $\alpha = 1.3,\ \beta = 0.1$



(d) Objective function $\mathcal{J}$ (in log-scale)

Figure 1: INDIAN applied to the nonsmooth function $\mathcal{J}(\theta_1, \theta_2) = 100(\theta_2 - |\theta_1|)^2 + |1 - \theta_1|$. Subplots (a-c) represent the trajectories of the parameters (θ_1, θ_2) in $\mathbb{R}^2$ for three choices of hyperparameters α and β (starting from the black dot). Subplot (d) displays the values of the objective function $\mathcal{J}(\theta_1, \theta_2)$ for the three settings considered.

INDIAN algorithm. Before addressing the nonsmooth stochastic case (see (4)), we present the smooth deterministic version: for an initial vector of parameters $\theta_0 \in \mathbb{R}^P$, we introduce an auxiliary variable $\psi_0 \in \mathbb{R}^P$, and iterate for $k \in \mathbb{N}$:

$$\text{(INDIAN)} \quad \begin{cases} \theta_{k+1} &= \theta_k + \gamma_k \left((\frac{1}{\beta} - \alpha)\theta_k - \frac{1}{\beta}\psi_k - \beta \nabla \mathcal{J}(\theta_k) \right) \\ \psi_{k+1} &= \psi_k + \gamma_k \left((\frac{1}{\beta} - \alpha)\theta_k - \frac{1}{\beta}\psi_k \right) \end{cases} \tag{1}$$

where $(\gamma_k)_{k\in\mathbb{N}}$ is a user-defined sequence of stepsizes (see Assumption 1 and Remark 2). This algorithm is an explicit discretization of an ordinary differential equation (ODE) introduced in [1] (see [6] for more details about discretization of ODEs).

$$\text{(DIN)} \quad \begin{cases} \dot{\theta}(t) + \beta \nabla \mathcal{J}(\theta(t)) + (\alpha - \frac{1}{\beta})\theta(t) + \frac{1}{\beta}\psi(t) = 0, \\ \dot{\psi}(t) + (\alpha - \frac{1}{\beta})\theta(t) + \frac{1}{\beta}\psi(t) = 0, \text{ a.e. on } (0, +\infty). \end{cases} \tag{2}$$

A notable fact is that the first coordinate θ of any bounded solution of (2) asymptotically converges to critical points of $\mathcal{J}$ [1]. Moreover, when $\mathcal{J}$ is twice continuously differentiable, (2) is equivalent to the following second order ODE strongly related to the laws of mechanics:

$$\underbrace{\ddot{\theta}(t)}_{\text{Inertial term}} + \underbrace{\alpha\,\dot{\theta}(t)}_{\text{Friction term}} + \underbrace{\beta\,\nabla^2 \mathcal{J}(\theta(t))\dot{\theta}(t)}_{\text{Newtonian effects}} + \underbrace{\nabla \mathcal{J}(\theta(t))}_{\text{Gravity effect}} = 0,\ t \geq 0. \tag{3}$$

The introduction of the twisted phase variable $\psi = -\beta\dot{\theta} - \beta^2 \nabla \mathcal{J}(\theta) - (\alpha\beta - 1)\theta$ in (2) allows to rewrite the second order dynamics (3) without explicitly involving the Hessian $\nabla^2 \mathcal{J}$. Equation (3) describes the movement of a mass point with coordinates $\theta(t)$ evolving on the landscape of the loss function $\mathcal{J}$: $\ddot{\theta}$ is the acceleration of the particle, $\nabla \mathcal{J}(\theta)$ represents a gravity effect, $\alpha\dot{\theta}$ corresponds to a viscous friction term and $\beta\,\nabla^2 \mathcal{J}(\theta)\dot{\theta}$ acts like a Newtonian damping. This analogy provides strong insights into the meaning of the hyperparameters α and β, which are damping coefficients allowing to control the trajectory and bring it to a flat valley of $\mathcal{J}$ (local or global minima). This feature is illustrated in Figure 1.

2 Theoretical guarantees of INDIAN for DL losses

A general structure for DL losses: Tameness. Tameness refers to a geometrical property satisfied by most finite-dimensional optimization problems met in practice, in particular in ML. Prominent classes of tame objects are piecewise linear objects, piecewise polynomial, or semi-algebraic objects but the notion is much more general: Sets or functions are called *tame* when they can be described by a finite number of basic formulas/inequalities/Boolean operations involving standard functions such as polynomial, exponential, or max functions. We refer to [2] for illustrations, recipes and examples within a general optimization setting or [11] for illustrations in the context of neural networks. One is referred to [10, 12, 21] for foundational material.

All finite-dimensional deep learning optimization models we are aware of yield tame losses $\mathcal{J}$. Consider for example a DNN with classical activation functions (ReLU, sigmoid, SQNL, tanh, soft plus, soft clipping, ...), and a standard dissimilarity function l such as ℓ_p norms or cross-entropy, then, the corresponding loss $\mathcal{J}$ is tame (see [11] for more examples). Hence tameness is a very mild assumption. It can be seen as a "non pathological" condition which is required to handle convergence analysis in the nonsmooth setting. The reader not comfortable with this notion may safely replace it by "ReLU network with square loss" to get a first understanding of our results.[3]

Handling the nonsmoothness of DL losses. The classical definition of the subgradient does not suit nonconvex nonsmooth functions (for example $x \mapsto -|x|$ has empty subgradient at 0). To overcome this problem we introduce the Clarke subdifferential: given a locally Lipschitz continuous function F between finite dimensional spaces, we define for each θ its Clarke subdifferential $\partial F(\theta)$ as the closed convex envelope of the limits of neighboring gradients (see [9] for a formal definition). This makes this set compact, convex and nonempty. Let us point out that the *classical* subgradient is included in the Clarke subgradient.

In order to compute the subdifferential of $\mathcal{J}$ and to cope with large datasets, $\mathcal{J}$ can be approximated by mini-batches, reducing the computational cost. However, unlike gradients, there is no sum-rule for Clarke subdifferentials (example: $0 = |\cdot| - |\cdot|$), which makes the traditional "*subgradient plus centered noise*" approach unfit to the study of mini-batch subsampling methods in DL. Thus we introduce a new notion, for any $\mathcal{B} \subset \{1, \dots, N\}$, set

$$\mathcal{J}_{\mathcal{B}} \colon \theta \mapsto \sum_{n \in \mathcal{B}} l(f(x_n, \theta), y_n), \quad D\mathcal{J}_{\mathcal{B}} = \sum_{n \in \mathcal{B}} \partial \left[l(f(x_n, \cdot), y_n) \right], \quad D\mathcal{J} = \sum_{n=1}^{N} \partial \left[(f(x_n, \cdot), y_n) \right].$$

Observe that, for each $\mathcal{B}$, we have $D\mathcal{J}_{\mathcal{B}} \supset \partial \mathcal{J}_{\mathcal{B}}$ and that $\mathcal{J}_{\mathcal{B}}$ is differentiable almost everywhere with $D\mathcal{J}_{\mathcal{B}} = \partial \mathcal{J}_{\mathcal{B}} = \{\nabla \mathcal{J}_{\mathcal{B}}\}$, see [9]. For convenience, a point satisfying $D\mathcal{J}(\theta) \ni 0$ will be called *D-critical*. This vocable is motivated by favourable properties whose statements and proofs are available in the extended version of this paper: a good calculus along curves and the existence of a nonsmooth Sard's theorem (this is where tameness plays a crucial role). To our knowledge, this notion has not previously been used in the literature. To model the mini-batch approximation, we consider a sequence $(\mathcal{B}_k)_{k \in \mathbb{N}}$ of nonempty subsets of $\{1, \dots, N\}$ chosen independently, uniformly at random with replacement and a sequence of positive stepsizes $(\gamma_k)_{k \in \mathbb{N}}$. In practice $v_k \in D\mathcal{J}_{\mathcal{B}_k}(\theta_k)$ is usually computed with a backpropagation algorithm, as in the seminal work of [20]. This results in a stochastic approximation of INDIAN applied deterministically to $\mathcal{J}$ (batch version). This can be seen by observing that the vectors v_k above may be written $\tilde{v}_k + \eta_k$, where $\tilde{v}_k \in D\mathcal{J}(\theta_k)$ and η_k compensates for the missing subgradients and can be seen as a zero-mean noise. Hence, INDIAN admits the following general abstract stochastic formulation:

$$(\text{INDIAN}_g) \begin{cases} \mu_{k+1} &= \mu_k + \gamma_k \left((\frac{1}{\beta} - \alpha)\mu_k - \frac{1}{\beta}\phi_k - \beta w_k + \xi_k \right) \\ \phi_{k+1} &= \phi_k + \gamma_k \left((\frac{1}{\beta} - \alpha)\mu_k - \frac{1}{\beta}\phi_k \right) \end{cases} \text{with } w_k \in D\mathcal{J}(\mu_k) \quad (4)$$

where $(\xi_k)_{k \in \mathbb{N}}$ is a martingale difference noise sequence adapted to the filtration induced by (random) iterates up to k, θ_0, ϕ_0 are arbitrary initial conditions, and $\alpha > 0$ and $\beta > 0$ are parameters of the algorithm (see Numerical experiments for example of choices of α and β).

Convergence results. To establish convergence[4], we start with the following standing assumption.
Assumption 1 (Vanishing stepsizes). *The stepsize sequence γ_k is positive, diverges ($\sum \gamma_k = +\infty$) and satisfies $\gamma_k = o\left(\frac{1}{\log k}\right)$, that is $\limsup_{k \to +\infty} |\gamma_k \log k| = 0$.*

[3]From now on we fix an o-minimal structure, an object is said to be tame if it belongs to this structure.

[4]All the proofs and theoretical tools are available in an extended version of this work [8].

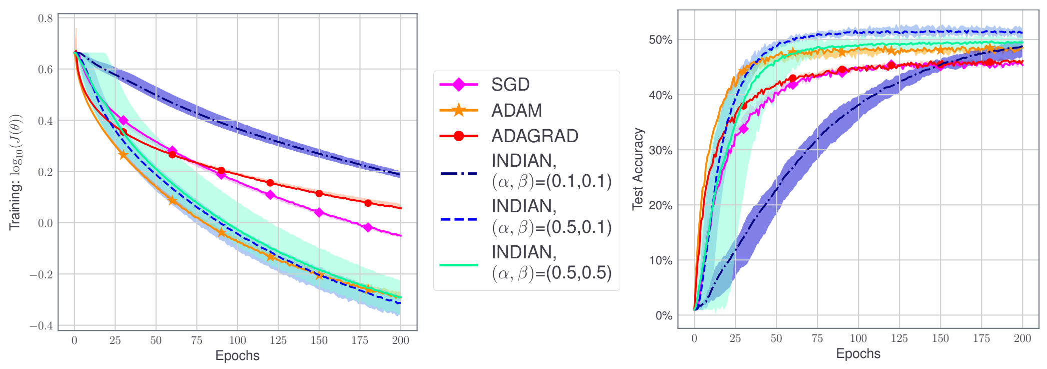


Figure 2: Optimization and accuracy results using NiN with CIFAR-100. Left: logarithm of the loss function $\mathcal{J}(\theta)$. Right: percentage accuracy on the test set. Solid lines represent mean values and pale surfaces represent the best and worst runs in terms of training loss and validation accuracy over five random initializations.

Typical admissible choices are $\gamma_k = C(k+1)^{-a}$ with $a \in (0,1]$, $C > 0$. The main theoretical result of this paper follows.

Theorem 1 (INDIAN$_g$ converges to the set of D-critical points of $\mathcal{J}$)**.** *Assume that $\mathcal{J}$ is locally Lipschitz continuous, tame and that the stepsizes satisfy Assumption 1. Let (θ_k, ψ_k) generated by* (4) *with an initial condition (θ_0, ψ_0) and assume that there exists $C > 0$ such that* $\sup_k \|(\theta_k, \psi_k)\| \le C$ *almost surely.*
Then, almost surely, any accumulation point $\bar{\theta}$ of a realization of the sequence $(\theta_k)_{k\in\mathbb{N}}$ satisfies $D\mathcal{J}(\bar{\theta}) \ni 0$. In addition $(\mathcal{J}(\theta_k))_{k\in\mathbb{N}}$ converges.

Remarks. (a) [Proof sketch:] The proof relies on the Lyapunov analysis of (3) in [1]. Tameness assumption provides nonsmooth calculus rules [11] and a nonsmooth Sard theorem [5], which allow to invoke the asymptotic analysis of stochastic approximation in [4].
(b) [Noise and Stepsizes]: Assumption 1 along with the finite sum structure of $\mathcal{J}$ allow much larger stepsizes than the usual Robbins-Monro condition [19], typically of the order $O(1/\sqrt{k})$, see [4, Remark 1.5] and [3] for more details. Other variants could be considered depending on the assumptions on the noise, see [4].
(c) [Convergence to critical points]: Observe that when $\mathcal{J}$ is differentiable, limit points are simply critical points. Moreover, let us mention that for general $\mathcal{J}$, being D-critical (or critical) is a necessary condition for being a local minimum.

3 Numerical Experiments

In this section, we use INDIAN for a concrete DL problem. We train a DNN for image classification using the CIFAR-100 dataset [15]. Regarding the network, we use a slightly modified version (with $P = 10^6$ parameters to optimize) of Network in Network (NiN) [17]. We compare our algorithm to the classical stochastic gradient descent (SGD) algorithm, ADAGRAD [13] and ADAM [14]. The full methodology can be found in the supplementary materials. We present the results for three different choices of hyperparameters of INDIAN using the intuition given by (3), even though there are many other satisfying choices of parameters. INDIAN is available as an optimizer for Pytorch, Keras and Tensorflow: `https://github.com/camcastera/Indian-for-DeepLearning/` [7].

Results. Our result are representative of what can be obtained with a moderately large network on CIFAR-100 with reasonable parameter tuning (though higher accuracy can be achieved by much larger networks). Fig. 2 (left) shows that ADAM and INDIAN (with adequate tuning) can outperform SGD and ADAGRAD for the training. Thus, INDIAN proves an efficient optimizer for this problem. Although ADAM's and INDIAN's performances are similar, the latter is more versatile since its hyperparameters are fully tunable compared to ADAM's adaptive stepsizes. Fig. 2 (right) highlights a special aspect of INDIAN. Indeed, every versions of INDIAN present better testing performances compared to the ones trained with usual optimizers, besides, the evolution of the accuracy depends on the choice of α and β. This suggests that the tuning of (α, β) might be a new regularization strategy in addition to usual methods such as dropout [22] and weight decay [16].

References

[1] Felipe Alvarez, Hedy Attouch, Jérôme Bolte, and Patrick Redont. A second-order gradient-like dissipative dynamical system with Hessian-driven damping: Application to optimization and mechanics. *Journal de Mathématiques Pures et Appliquées*, 81(8):747–779, 2002.

[2] Hédy Attouch, Jérôme Bolte, Patrick Redont, and Antoine Soubeyran. Proximal alternating minimization and projection methods for nonconvex problems: An approach based on the Kurdyka-Łojasiewicz inequality. *Mathematics of Operations Research*, 35(2):438–457, 2010.

[3] Michel Benaïm. Dynamics of stochastic approximation algorithms. In *Séminaire de Probabilités XXXIII*, pages 1–68. Springer, 1999.

[4] Michel Benaïm, Josef Hofbauer, and Sylvain Sorin. Stochastic approximations and differential inclusions. *SIAM Journal on Control and Optimization*, 44(1):328–348, 2005.

[5] Jérôme Bolte, Aris Daniilidis, Adrian Lewis, and Masahiro Shiota. Clarke subgradients of stratifiable functions. *SIAM Journal on Optimization*, 18(2):556–572, 2007.

[6] John C. Butcher. *Numerical methods for ordinary differential equations*. John Wiley & Sons, 2016.

[7] Camille Castera. INDIAN for deep learning. `https://github.com/camcastera/Indian-for-DeepLearning/`, 2019.

[8] Camille Castera, Jérôme Bolte, Cédric Févotte, and Edouard Pauwels. An inertial newton algorithm for deep learning. *arXiv preprint arXiv:1905.12278*, 2019.

[9] Frank H. Clarke. *Optimization and nonsmooth analysis*. SIAM, 1990.

[10] Michel Coste. *An introduction to o-minimal geometry*. Istituti editoriali e poligrafici internazionali Pisa, 2000.

[11] Damek Davis, Dmitriy Drusvyatskiy, Sham Kakade, and Jason D. Lee. Stochastic subgradient method converges on tame functions. *Foundations of Computational Mathematics*, 2019. (in press).

[12] Lou van den Dries. *Tame topology and o-minimal structures*. Cambridge university press, 1998.

[13] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(7):2121–2159, 2011.

[14] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv:1412.6980*, 2014.

[15] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, Canadian Institute for Advanced Research, 2009.

[16] Anders Krogh and John A. Hertz. A simple weight decay can improve generalization. In *Advances in neural information processing systems*, pages 950–957, 1992.

[17] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.

[18] Boris T. Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964.

[19] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, 22(1):400–407, 1951.

[20] David E. Rumelhart and Geoffrey E. Hinton. Learning representations by back-propagating errors. *Nature*, 323(9):533–536, 1986.

[21] Masahiro Shiota. *Geometry of subanalytic and semialgebraic sets*, volume 150. Springer Science & Business Media, 2012.

[22] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

A Full Methodology of Section 3

We train a DNN for image classification using the CIFAR-100 dataset [15]. This dataset is composed of $60,000$ small colored images of size $32 \times 32 \times 3$ each associated with a label (airplane, cat, etc.). We split the dataset into $50,000$ images for the training procedure and use the $10,000$ other images to validate the model. Regarding the network, we use a sightly modified version of Network in Network of [17]. It is a very popular network that consists of 2D-convolutional layers with max-pooling and dense layers with ReLU activation functions, our version has 1 million parameters to optimize. The loss function used is the categorical cross-entropy. We compare our algorithm to the classical stochastic gradient descent (SGD) algorithm, and both the very popular ADAGRAD [13] and ADAM [14] algorithms. At each iteration, we compute the approximation of $\partial \mathcal{J}(\theta)$ on a subset $\mathcal{B} \subset \{1, \ldots, 50000\}$ of size 32. To do a fair comparison, each algorithm is initialized with the same random weights (following a normal distribution). To get relevant results, this process is done for five different random initializations of the network.

The sequence of steps γ_k has to meet Assumption 1 and we chose the classical schedule $\gamma_k = \frac{\gamma_0}{\sqrt{k+1}}$ for both INDIAN and SGD, where γ_0 is the initial stepsize. Regarding ADAGRAD and ADAM, starting from γ_0 they have there own adaptive procedure to generate $(\gamma_k)_{k\in\mathbb{N}}$, see [13, 14] for more details. For all four algorithms, choosing the right initial step length γ_0 is often critical in terms of efficiency. We use a grid-search for each algorithm and chose the initial stepsize that most decreases the loss function over ten epochs (ten complete passes over the data). Then we run each algorithm during 200 epochs with a batchsize of size 32.