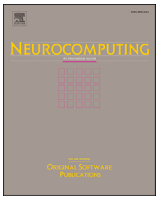
Neurocomputing 658 (2025) 131738

Contents lists available at ScienceDirect

Neurocomputing

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



Exact Gauss-Newton optimization for training deep neural networks

Mikalai Korbit*, Adeyemi D. Adeoye, Alberto Bemporad, Mario Zanon

IMT School for Advanced Studies Lucca, Lucca, Italy

ARTICLE INFO

Communicated by J. Na

Keywords:
Stochastic optimization
Second-order optimization
Gauss-newton hessian approximation
Machine learning
Reinforcement learning

ABSTRACT

We present Exact Gauss-Newton (EGN), a stochastic second-order optimization algorithm that combines the generalized Gauss-Newton (GN) Hessian approximation with low-rank linear algebra to compute the descent direction. Leveraging the Duncan-Guttman matrix identity, the parameter update is obtained by factorizing a matrix which has the size of the mini-batch. This is particularly advantageous for large-scale machine learning problems where the dimension of the neural network parameter vector is several orders of magnitude larger than the batch size. Additionally, we show how improvements such as line search, adaptive regularization, and momentum can be seamlessly added to EGN to further accelerate the algorithm. Moreover, under mild assumptions, we prove that our algorithm converges in expectation to a stationary point of the objective. Finally, our numerical experiments demonstrate that EGN consistently exceeds, or at most matches the generalization performance of well-tuned SGD, Adam, GAF, SQN, and SGN optimizers across various supervised and reinforcement learning tasks.

1. Introduction

Optimization plays a pivotal role in Machine Learning (ML), with gradient-based methods being at the forefront. Stochastic Gradient Descent (SGD) [1], a first-order stochastic optimization algorithm, and its accelerated versions such as momentum-based approaches [2–5], adaptive learning rates [6–8], and a combination of the two [9–11], have been instrumental in numerous ML applications. For example, in Computer Vision (CV) ResNets [12] are trained with SGD, AdaGrad [6] is used for training recommendation systems [13], language models GPT-3 [14] and LLaMA [15] are optimized with Adam [9] and AdamW [16], respectively. Despite their cheap and relatively easy-to-implement updates, first order-methods (FOMs) suffer from several shortcomings. FOMs are sensitive to hyper-parameter selection, and the optimal hyper-parameter set typically does not transfer well across different problems which leads to a costly procedure of hyper-parameter tuning. Also, FOMs are slow to converge in the flat regions of the loss landscape, where the Hessian is ill-conditioned [17].

Second-order methods (SOMs) incorporate the (approximate) curvature information into the update in order to effectively precondition the gradient vector. In contrast to first-order algorithms, SOMs are shown to be robust to the selection of hyper-parameters [18] and to potentially offer faster convergence [19,20]. So far, the adoption of second-order methods for ML problems has been limited due to the complexity of calculating and storing the Hessian and the computational load of solving the linear system $\mathbf{Hd} = -\mathbf{g}$, where $\mathbf{H}$ is the (approximate) Hessian matrix, $\mathbf{g}$ is the gradient of the loss function, and $\mathbf{d}$ is the descent direction. Addressing these computational challenges, most approaches use a combination of Hessian approximation and an efficient algorithmic technique for solving the linear system. Common approximations to the Hessian include diagonal scaling [21,22], the empirical Fisher matrix [23], the quasi-Newton approach [24–26], and the Gauss-Newton (GN) approximation [27–29].

In this work, we follow the Gauss-Newton approach to Hessian approximation. We apply a special derivation inspired by [44,45] which uses an efficient exact linear algebra identity—the Duncan-Guttman (DG) formula [30,31]—to speed up the inversion of the Hessian matrix. Compared to Hessian-free optimization (HFO) [32–34] and Inexact Gauss-Newton (iGN) [28], this approach allows Exact Gauss-Newton (EGN) to solve the system $\mathbf{Hd} = -\mathbf{g}$ *exactly* with the same algorithmic complexity burden. Moreover, compared to methods that directly apply the Sherman-Morrison-Woodbury (SMW) formula (see, e.g., [23]), we solve for the descent direction in fewer matrix operations, thus reducing the algorithmic complexity.

Our contributions are as follows.

* Corresponding author.
Email address: mikalai.korbit@imtlucca.it (M. Korbit).

https://doi.org/10.1016/j.neucom.2025.131738
Received 29 April 2025; Received in revised form 25 July 2025; Accepted 4 October 2025
Available online 9 October 2025

- We propose the EGN algorithm, which relies on a regularized Gauss-Newton Hessian matrix and exploits the Duncan-Guttman identity to efficiently solve the linear system.
- We provide a theoretical analysis of the EGN algorithm and establish that EGN finds a stationary point in expectation for a large enough iteration count.
- We evaluate the performance of EGN on several supervised learning and reinforcement learning tasks using various neural network architectures.

2. Related work

Our method can be viewed within the broader context of approximate second-order stochastic optimization. Some notable approaches include diagonal scaling [21,22], Krylov subspace descent [35], Hessian-free optimization [32–34], quasi-Newton approaches [24–26,36,37], Gauss-Newton [27,38] and Natural Gradient [39,40] methods. A detailed overview of second-order optimization methods for large-scale machine learning problems can be found in [18,41,42].

Most closely related to our work are the algorithms inspired by the Gauss-Newton approach. Such methods approximate the Hessian of the loss function using only first-order sensitivities. In practice, the damped version of the Gauss-Newton direction is often calculated, forming the stochastic Levenberg-Marquardt (SLM) group of algorithms. We can classify SLM methods by three dimensions: (a) by the type of the Jacobian estimation algorithm used; (b) by the matrix inversion algorithm; and (c) by additional adaptive parameters and acceleration techniques. Based on this paradigm we summarize selected SLM algorithms in Table 1.

The Jacobian can either be calculated exactly through the reverse mode of automatic differentiation as proposed by, e.g., [27–29] or be estimated approximately. Low rank Jacobian estimation is suggested by the NLLS1 and NLLSL algorithms [29] with experimental results showing almost on par performance with the exact Jacobian version of the methods. SGN2 [28] uses SARAH estimators for approximating function values and Jacobians with SGN2 performing better than SGN [28] that assumes the exact Jacobian. In [22] the Gauss-Newton-Bartlett (GNB) estimator is introduced to adapt the GN method to large-scale classification problems. EGN does not mandate a specific computation technique for the Jacobian. In our experiments we rely on backpropagation deferring other methods to further research.

Solving $\mathbf{Hd} = -\mathbf{g}$ naively for a neural network with d parameters has complexity $\mathcal{O}\left(d^3\right)$. The procedure becomes practically infeasible even for networks of moderate size, so several alternative approaches have been proposed. Following [20], we distinguish between *inexact* and *exact* solutions to the linear system. Inexact methods rely on iterative algorithms to solve the system approximately in as few iterations as possible. Among such algorithms we mention the Conjugate Gradient (CG) method used in Hessian-free optimization [33,34] as well as in GN methods like SGN [27] and LM [43]; the Accelerated Dual Proximal-Gradient (ADPG) method proposed in [28] for SGN and SGN2 solvers; the Stochastic Gradient Iteration approach (SGI), analysed in [20] as part of the Newton-SGI solver. Exact methods are typically based on linear algebra identities. For example, in [23,29] the system is solved exactly with the Sherman-Morrison-Woodbury (SMW) formula. Contrarily, we follow [45] and derive the EGN update formula using the Duncan-Guttman matrix identity [30,31].

State-of-the-art implementations of the GN algorithm often include additional improvements to address the issues of stochasticity of the Hessian matrix, computational load of calculating the Jacobian and adaptive hyper-parameter tuning. Just as with the gradient, the stochastic sampling introduces noise in the Hessian which leads to the erroneous descent direction. A common solution to combat noisy estimates is to add temporal averaging (or momentum), e.g., with exponential moving averages (EMA). Examples of such approach are AdaHessian [21] and Sophia [22] that keep EMA of a diagonal Hessian, as well as [34] that incorporates momentum into the HFO framework. The idea of reusing the Hessian estimate from previous iterations is formalized in [46] showing that evaluating the Hessian "lazily" once per k iterations significantly reduces the computational burden while at the same time does not degrade the performance that much. Although second-order methods typically require less tuning [18,27,38], the SLM approach still requires setting a learning rate α and the regularization parameter λ. The line search for α, common in the deterministic optimization, is problematic in the stochastic setting due to the high variance of the loss gradient norm [47]. Still, there are promising attempts to incorporate line search into quasi-Newton methods [36], HFO [34] as well as Gauss-Newton [43]. Adaptive regularization in a manner similar to the deterministic Levenberg-Marquardt approach is proposed in [23,48].

3. Preliminaries

We use boldface letters to denote vectors and matrices. The $n \times n$ identity matrix is denoted by $\mathbf{I}_n$, and we omit the subscript when the size is clear from the context. The subscript t represents the iteration within the optimization loop, and may be omitted to avoid overloading the notation. We define the standard inner product between two vectors $\mathbf{x}, \mathbf{y}$ as $\langle \mathbf{x}, \mathbf{y} \rangle := \mathbf{x}^\top \mathbf{y}$. The standard Euclidean norm is denoted by $\|\cdot\|$, and the expected value of a random variable is denoted by $\mathbb{E}[\cdot]$.

We adopt the Empirical Risk Minimization (ERM) framework [49] and consider the problem of finding the weights $\mathbf{w} \in \mathbb{R}^d$ of a parametric function $\Phi : \mathbb{R}^m \times \mathbb{R}^d \to \mathbb{R}^c$, e.g., a neural network, such that it minimizes the empirical risk over a dataset $\mathcal{D}$ consisting of N pairs $(\mathbf{y}_i, \mathbf{x}_i)$ where $\mathbf{x}_i \in \mathbb{R}^m$ is a vector of features and $\mathbf{y}_i \in \mathbb{R}^c$, $c \geq 1$ is a target vector. We want to solve the following optimization problem:

$$\mathbf{w}^* \in \arg\min_{\mathbf{w} \in \mathbb{R}^d} \mathcal{L}_N(\mathbf{w}) := \mathbb{E}_{\xi \sim P_\xi}[\mathcal{L}_N(\mathbf{w}; \xi)], \tag{1}$$

where ξ is a random variable with distribution P_ξ. The objective function is expressed as a finite-sum of functions $\mathcal{L}_N(\mathbf{w}; \xi_i)$ over the realization $\mathcal{D}$ of ξ, and

$$\mathcal{L}_N(\mathbf{w}) := \frac{1}{N} \sum_{i=1}^{N} \ell\left(\mathbf{y}_i, \Phi(\mathbf{x}_i; \mathbf{w})\right), \tag{2}$$

is the empirical risk with $\ell : \mathbb{R}^c \times \mathbb{R}^c \to \mathbb{R}$ – a loss function, involving a single pair $\left(\mathbf{y}_i, \mathbf{x}_i\right)$ only.

We assume all these functions to be twice differentiable with respect to $\mathbf{w}$. Note that, while this assumption could be partially relaxed, we stick to it for the sake of simplicity.

Table 1
A survey on Gauss-Newton methods for large-scale stochastic optimization.

Algorithm	Jacobian estimation	Solving the linear system	Additional improvements
SGN [27]	Exact via reverse mode autodiff	Approximate with CG	–
LM [43]	Exact via reverse mode autodiff	Approximate with CG	Line search, momentum, uphill step acceptance
SGN [28]	Exact via reverse mode autodiff	Approximate with ADPG	–
SGN2 [28]	Approximate with SARAH estimators	Approximate with ADPG	–
NLLS1, NLLSL [29]	Rank-1, Rank-L approximation	Exact with SMW formula	–
SMW-GN [23]	Exact via reverse mode autodiff	Exact with SMW formula	Adaptive regularization
EGN (this paper)	Any Jacobian estimation algorithm (exact via backpropagation is the default)	Exact with DG identity (see Theorem 4.1 and Lemma 4.2)	Line search, adaptive regularization, momentum

3.1. Gradient-based optimization

Problem (1) is typically solved by variations of the Stochastic Gradient Descent (SGD) method by sampling mini-batches $\mathcal{B}_t$ from $\mathcal{D}$ rather than processing the entire dataset in each iteration. We denote the loss on a mini-batch $\mathcal{L}_b$ as

$$\mathcal{L}_b(\mathbf{w}) := \frac{1}{b}\sum_{i=1}^{b} \ell\left(\mathbf{y}_i, \Phi(\mathbf{x}_i;\mathbf{w})\right), \tag{3}$$

where b is the batch size. The iterations take the form

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \alpha_t \mathbf{d}_t, \tag{4}$$

where $\alpha_t > 0$ is a learning rate and $\mathbf{d}_t$ is a descent direction obtained from the gradient. In general, we have $\mathbf{d}_t = -\mathbf{C}_t\mathbf{g}_t$ where $\mathbf{C}_t$ is a preconditioning matrix that scales, rotates, and shears the gradient of the mini-batch loss $\mathbf{g} := \nabla_{\mathbf{w}}\mathcal{L}_b$. Notice that by setting $b = 1$, $\mathbf{C}_t = \mathbf{I}_d$ and $\mathbf{d}_t = -\mathbf{g}_t$ we recover the incremental SGD update [1]. In practice, the preferred training algorithm is often an accelerated version of mini-batch SGD.

Minimizing the quadratic approximation of the batch loss leads to the following linear system

$$\mathbf{H}_t\mathbf{d}_t = -\mathbf{g}_t, \tag{5}$$

where $\mathbf{H}_t := \nabla^2_{\mathbf{w}}\mathcal{L}_b$ is the Hessian matrix of the mini-batch loss. Finding the direction $\mathbf{d}_t$ by solving the system (5) using $\mathbf{H}_t$ or its approximation defines the broad spectrum of second-order methods. Setting $\mathbf{C}_t = \mathbf{H}_t^{-1}$ results in Newton's method. This method suffers from several drawbacks: (a) one needs to compute second-order derivatives with respect to $\mathbf{w}$; (b) $\mathbf{C}_t$ has to be positive-definite to ensure descent; (c) the linear system (5) must be solved, which in general scales cubically with the dimension of $\mathbf{w}$; and (d) since $\mathbf{H}_t$ is a noisy estimate of the true Hessian of the empirical risk $\mathcal{L}$, $\mathbf{H}_t^{-1}$ can result in suboptimal conditioning. These issues and the fact that $\mathbf{w}$ is of rather large dimension have made the direct application of Newton's method practically irrelevant in ML applications. Indeed, for problems like Large Language Model (LLM) pre-training [14,15,50] or CV tasks [12], $\mathbf{w}$ can be extremely high-dimensional, e.g., $314 \cdot 10^9$ parameters for Grok-1 [51] model, $8 \cdot 10^9$ to $405 \cdot 10^9$ parameters for LLaMA-family models [52] and $0.27 \cdot 10^6$ to $19.4 \cdot 10^6$ parameters for ResNets [12], which explains why accelerated first-order methods are usually preferred.

In order to make SOMs scalable for ML applications one could instead approximate the inverse Hessian, i.e., $\mathbf{C}_t \approx \mathbf{H}_t^{-1}$. Examples of such preconditioning include diagonal scaling (e.g., with Hutchinson method [21,22]) with the idea of extracting the (approximate) diagonal elements of $\mathbf{H}_t$ while neglecting the off-diagonal terms; the quasi-Newton approach [25,26] that approximates the Hessian using the information from past and current gradients; and the Gauss-Newton method [23,27] that leverages the Jacobian of the residuals, neglecting second-order cross-derivatives, particularly suitable for loss functions structured as (2). In this paper, we use the Gauss-Newton Hessian approximation which is shown to provide a good approximation of the true Hessian for ML applications [17,53].

3.2. Generalized gauss-newton hessian approximation

We consider the Generalized Gauss-Newton (GGN) Hessian approximation scheme [41,53,54] which is suited for both regression and multi-class classification tasks. The GGN Hessian approximation is constructed using only first-order sensitivities (see the derivation in Appendix A) to obtain

$$\mathbf{H}^{\text{GN}} = \frac{1}{b}\mathbf{J}^\top\mathbf{Q}\mathbf{J} \tag{6}$$

where we vertically stack individual Jacobians $\mathbf{J}_{\Phi_i} := \frac{\partial\Phi(\mathbf{x}_i;\mathbf{w})}{\partial\mathbf{w}}$ for each sample in the batch $\mathcal{B}$ to form $\mathbf{J} = \begin{bmatrix} \mathbf{J}_{\Phi_1} & \dots & \mathbf{J}_{\Phi_b} \end{bmatrix}^\top \in \mathbb{R}^{bc\times d}$ and construct a block diagonal matrix $\mathbf{Q} = \text{blkdiag}(\mathbf{Q}_{\ell_1}, \mathbf{Q}_{\ell_2}, \dots, \mathbf{Q}_{\ell_b}) \in \mathbb{R}^{bc\times bc}$, where $\mathbf{Q}_{\ell_i} = \frac{\partial^2\ell(\mathbf{y}_i,\Phi(\mathbf{x}_i;\mathbf{w}))}{\partial\Phi^2} \in \mathbb{R}^{c\times c}$. As pointed out in [55], approximation (6) is justified since the true Hessian is dominated by the term $\mathbf{J}^\top\mathbf{Q}\mathbf{J}$. Using the inverse of the GGN Hessian as a preconditioner yields the following direction

$$\mathbf{d}_t^{\text{GN}} = -\left(\frac{1}{b}\mathbf{J}_t^\top\mathbf{Q}_t\mathbf{J}_t\right)^{-1}\mathbf{g}_t, \tag{7}$$

which, by defining $\Delta\mathbf{w} = \mathbf{w} - \mathbf{w}_t$, corresponds to the solution of the quadratic program

$$\mathbf{d}_t^{\text{GN}} = \arg\min_{\Delta\mathbf{w}} \frac{1}{2}\Delta\mathbf{w}^\top \underbrace{\frac{1}{b}\mathbf{J}_t^\top\mathbf{Q}_t\mathbf{J}_t}_{\mathbf{H}_t^{\text{GN}}}\Delta\mathbf{w} + \mathbf{g}_t^\top\Delta\mathbf{w}. \tag{8}$$

The Gauss-Newton step (7) solves issues (a) and, partially, (b), since no second-order derivatives need to be computed and the Hessian approximation is positive semi-definite by construction provided that the loss function $\ell\left(\mathbf{y}_i, \Phi(\mathbf{x}_i;\mathbf{w})\right)$ is convex. A positive-definite Hessian approximation can easily be obtained from $\mathbf{H}^{\text{GN}}$, e.g., by adding to it a small constant times the identity matrix. This approach is called Levenberg-Marquardt (LM) [56] and is often used in practice, such that

$$\mathbf{H}^{\text{LM}} = \frac{1}{b}\mathbf{J}^\top\mathbf{Q}\mathbf{J} + \lambda\mathbf{I}_d \tag{9}$$

with $\lambda > 0$. The regularizer $\lambda\mathbf{I}_d$ serves a dual purpose: it ensures that the approximate Hessian matrix is invertible and also helps to avoid over-fitting [49].

The Gauss-Newton update, however, still potentially suffers from issue (c), i.e., the need to solve a linear system, which can be of cubic complexity, and (d) the noise in the Hessian estimate. Since issue (c) is a centerpiece of our method, and issue (d) is an inherent part of stochastic sampling we defer the discussion on both of them to Section 4.

We conclude by examining the GGN update in two common machine learning tasks—regression and multi-class classification—and how it is derived in each case.

Regression. Regression is a predictive modeling task where the objective is to predict a scalar numeric target. For regression tasks we define the loss function ℓ as the mean squared error (MSE)

$$\ell\left(\mathbf{y}_i, \Phi(\mathbf{x}_i;\mathbf{w})\right) := \frac{1}{2}\left(\Phi(\mathbf{x}_i;\mathbf{w}) - \mathbf{y}_i\right)^2. \tag{10}$$

We can show that the gradient and the GN Hessian for the MSE loss read

$$\mathbf{g} = \frac{1}{b}\mathbf{J}^\top\mathbf{r}, \qquad \mathbf{H}^{\text{GN}} = \frac{1}{b}\mathbf{J}^\top\mathbf{J}, \tag{11}$$

where $\mathbf{J} \in \mathbb{R}^{b\times d}$ is a matrix of stacked Jacobians, and $\mathbf{r} \in \mathbb{R}^b$ is a vector of residuals defined as $\mathbf{r} := \begin{bmatrix} \Phi(\mathbf{x}_1;\mathbf{w}) - \mathbf{y}_1 & \dots & \Phi(\mathbf{x}_b;\mathbf{w}) - \mathbf{y}_b \end{bmatrix}^\top$.

Multi-class classification. The task of multi-class classification is to predict a correct class from c classes given a vector of features $\mathbf{x}_i$. For such problems the output of the neural network Φ is a c-dimensional vector of prediction scores (logits) $\mathbf{z}_i = \Phi(\mathbf{x}_i;\mathbf{w})$ and the target vector is a one-hot encoded vector $\mathbf{y}_i = e_k$, where e_k denotes column k of the identity matrix $\mathbf{I}_c$ and k is the index of the correct class. We define the loss function as a softmax cross-entropy loss (CE)

$$\ell\left(\mathbf{y}_i, \Phi(\mathbf{x}_i;\mathbf{w})\right) := -\sum_{k=1}^{c} \mathbf{y}_{i,k}\log\left(\sigma\left(\mathbf{z}_{i,k}\right)\right), \tag{12}$$

where $\sigma(\mathbf{z}_{i,k}) = \frac{e^{\mathbf{z}_{i,k}}}{\sum_{j=1}^{c} e^{\mathbf{z}_{i,j}}}$, $\mathbf{y}_{i,k}$ and $\mathbf{z}_{i,k}$ are k-th elements of vectors $\mathbf{y}_i$ and $\mathbf{z}_i$ respectively.

The gradient and the GN Hessian for the CE loss are

$$\mathbf{g} = \frac{1}{b}\mathbf{J}^\top \mathbf{r}, \qquad \mathbf{H}^{\text{GN}} = \frac{1}{b}\mathbf{J}^\top \mathbf{Q}\mathbf{J}, \tag{13}$$

where $\mathbf{J} \in \mathbb{R}^{bc\times d}$ is a matrix of stacked Jacobians, $\mathbf{r} \in \mathbb{R}^{bc}$ is a vector of (pseudo-)residuals defined as $\mathbf{r} := \begin{bmatrix} \left(\sigma\left(\Phi(\mathbf{x}_1;\mathbf{w})\right) - \mathbf{y}_1\right)^\top & \dots & \left(\sigma\left(\Phi(\mathbf{x}_b;\mathbf{w})\right) - \mathbf{y}_b\right)^\top \end{bmatrix}^\top$, and $\mathbf{Q} \in \mathbb{R}^{bc\times bc}$ is a block diagonal matrix of stacked matrices $\mathbf{Q}_{\ell_i}$ that each have $\sigma(\mathbf{z}_{i,k})(1-\sigma(\mathbf{z}_{i,k}))$ across the diagonal and $-\sigma(\mathbf{z}_{i,k})\sigma(\mathbf{z}_{i,l})$ off-diagonal.

4. Algorithm

We are now ready to present the EGN algorithm. First, we will discuss how one can efficiently solve the linear system. Then, we will discuss further enhancements to the basic algorithm. Next, we address issue (c), i.e., the problem of finding the solution of the symmetric linear system $\mathbf{H}_t\mathbf{d}_t = -\mathbf{g}_t$.

Substituting the exact Hessian with the regularized Gauss-Newton Hessian yields

$$\left(\frac{1}{b}\mathbf{J}_t^\top \mathbf{Q}_t \mathbf{J}_t + \lambda_t \mathbf{I}_d\right)\mathbf{d}_t^{\text{LM}} = -\frac{1}{b}\mathbf{J}_t^\top \mathbf{r}_t, \tag{14}$$

where for the MSE loss $c = 1$ and $\mathbf{Q}_t = \mathbf{I}_b$. Solving (14) for $\mathbf{d}_t^{\text{LM}}$ requires one to factorize matrix $\mathbf{H}^{\text{LM}}$, carrying a complexity of $\mathcal{O}\left(d^3\right)$. We notice, however, that in practice one often has $d \gg bc$, i.e., the parameter vector is of very high dimension, e.g., $d > 10^6$. In that case, the GN Hessian matrix (6) is low-rank by construction. That allows us to transfer the computationally expensive inversion operation from the high-dimensional $d \times d$ space (which is the original dimension of the Hessian) to the low-dimensional $bc \times bc$ space.

To that end, we follow an approach similar to the ones in [44,45] which propose to utilize the Duncan-Guttman identity [30,31]. We present this in Theorem 4.1 and Lemma 4.2 below for the Levenberg-Marquardt direction. For a general presentation which considers smooth regularization functions, we refer the interested reader to [45, Section 3].

Theorem 4.1 ([30,31]). *Assuming* $\mathbf{A}$ *and* $\mathbf{D}$ *are full-rank matrices, the following identity holds*

$$\left(\mathbf{A} - \mathbf{B}^\top \mathbf{D}^{-1}\mathbf{C}\right)^{-1}\mathbf{B}^\top \mathbf{D}^{-1} = \mathbf{A}^{-1}\mathbf{B}^\top\left(\mathbf{D} - \mathbf{C}\mathbf{A}^{-1}\mathbf{B}^\top\right)^{-1}. \tag{15}$$

By observing that Eq. (14) defining the direction $\mathbf{d}^{\text{LM}}$ has the form of the left-hand side of the identity (15) we state the following.

Lemma 4.2. *The Levenberg-Marquardt direction* $\mathbf{d}$ *(Eq. (14)) for both MSE and CE loss functions can be computed using Algorithm 1.*

Proof. Substituting $\mathbf{A} = b\lambda\mathbf{I}_d$, $\mathbf{B}^\top = -\mathbf{J}^\top$, $\mathbf{C} = \mathbf{J}$ and $\mathbf{D} = \mathbf{Q}^{-1}$ into Eq. (15) we get

$$\left(b\lambda\mathbf{I}_d + \mathbf{J}^\top\mathbf{Q}\mathbf{J}\right)^{-1}\mathbf{J}^\top\mathbf{Q} = -\left(b\lambda\mathbf{I}_d\right)^{-1}\mathbf{J}^\top\left(\mathbf{Q}^{-1} + \mathbf{J}\left(b\lambda\mathbf{I}_d\right)^{-1}\mathbf{J}^\top\right)^{-1} \tag{16}$$

Multiplying both sides by $\mathbf{Q}^{-1}\mathbf{r}$ yields

$$\left(b\lambda\mathbf{I}_d + \mathbf{J}^\top\mathbf{Q}\mathbf{J}\right)^{-1}\mathbf{J}^\top\mathbf{r} = -\left(b\lambda\mathbf{I}_d\right)^{-1}\mathbf{J}^\top\left(\mathbf{Q}^{-1} + \mathbf{J}\left(b\lambda\mathbf{I}_d\right)^{-1}\mathbf{J}^\top\right)^{-1}\mathbf{Q}^{-1}\mathbf{r} \tag{17}$$

where the lhs of (17) is now equivalent to explicitly solving for $\mathbf{d}^{\text{LM}}$ in Eq. (14). Simplifying the rhs of (17) results in

$$\mathbf{d}^{\text{LM}} = -\mathbf{J}^\top\left(b\lambda\mathbf{Q}^{-1} + \mathbf{J}\mathbf{J}^\top\right)^{-1}\mathbf{Q}^{-1}\mathbf{r}. \tag{18}$$

Applying the inverse of a product property, $\mathbf{B}^{-1}\mathbf{A}^{-1} = (\mathbf{A}\mathbf{B})^{-1}$, to the expression $\left(b\lambda\mathbf{Q}^{-1} + \mathbf{J}\mathbf{J}^\top\right)^{-1}\mathbf{Q}^{-1}$ we have

$$\left(b\lambda\mathbf{Q}^{-1} + \mathbf{J}\mathbf{J}^\top\right)^{-1}\mathbf{Q}^{-1} = \left(\mathbf{Q}\left(b\lambda\mathbf{Q}^{-1} + \mathbf{J}\mathbf{J}^\top\right)\right)^{-1} = \left(b\lambda\mathbf{I}_{bc} + \mathbf{Q}\mathbf{J}\mathbf{J}^\top\right)^{-1}. \tag{19}$$

So that the direction can be calculated as

$$\mathbf{d}^{\text{LM}} = -\mathbf{J}^\top\left(\mathbf{Q}\mathbf{J}\mathbf{J}^\top + b\lambda\mathbf{I}_{bc}\right)^{-1}\mathbf{r}, \tag{20}$$

which corresponds to the procedure outlined in Algorithm 1. □

4.1. Comparison to existing methods

The key property of Algorithm 1 is that the system (5) is solved *exactly* in contrast to approximate (or *inexact*) solutions that underpin algorithms such as HFO [32,34], Newton-SGI [20], LiSSA [19], SGN [27] and iGN [28]. The benefits of having the exact solution are analyzed in [20] with exact Newton methods enjoying faster convergence rates than inexact ones. Our experiments (Section 6) also demonstrate that the exact Gauss-Newton solver (EGN) consistently outperforms the inexact version (SGN) across the majority of the problems. The complexity of Algorithm 1 is dominated by the matrix multiplication $\mathbf{J}\mathbf{J}^\top$ that costs $\mathcal{O}\left(b^2\,c^2\,d\right)$ as well as solving the linear system of size $bc \times bc$ with complexity $\mathcal{O}\left(b^3\,c^3\right)$. Assuming $d > bc$, the overall complexity of Algorithm 1 is $\mathcal{O}\left(b^2\,c^2\,d\right)$.

A common alternative way to solve the Levenberg-Marquardt linear system (14) exactly is to apply the SMW identity, as in [23,29,48,57]. The SMW identity states

$$(\mathbf{A} + \mathbf{U}\mathbf{C}\mathbf{V})^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{U}\left(\mathbf{C}^{-1} + \mathbf{V}\mathbf{A}^{-1}\mathbf{U}\right)^{-1}\mathbf{V}\mathbf{A}^{-1}. \tag{21}$$

With $\mathbf{A} = \lambda\mathbf{I}_d$, $\mathbf{U} = \mathbf{J}^\top$, $\mathbf{C} = \frac{1}{b}\mathbf{Q}$, and $\mathbf{V} = \mathbf{J}$, we obtain the SMW-GN matrix inversion [23]:

$$\left(\frac{1}{b}\mathbf{J}^\top\mathbf{Q}\mathbf{J} + \lambda\mathbf{I}_d\right)^{-1} = \frac{1}{\lambda}\mathbf{I}_d - \frac{1}{\lambda^2}\mathbf{J}^\top\left(b\mathbf{Q}^{-1} + \frac{1}{\lambda}\mathbf{J}\mathbf{J}^\top\right)^{-1}\mathbf{J}. \tag{22}$$

As with EGN, the dominant term is $\mathbf{J}\mathbf{J}^\top$. However, even with efficient ordering of operations, there are at least two $\mathcal{O}\left(b^2\,c^2\,d\right)$ matrix multiplications while EGN requires just one. Beyond doubling the per-step cost, the additional multiplication can introduce extra rounding error that gradually accumulates [58].

Another important consideration is the non-regularized case. When Levenberg damping is disabled ($\lambda = 0$), $\mathbf{A}$ in (21) is singular and the SMW inverse is undefined. EGN remains well-posed provided $\mathbf{Q}\mathbf{J}\mathbf{J}^\top$ is invertible, making it applicable to pure Gauss-Newton steps.

We support our theoretical findings with the empirical comparison of solving Eq. (14) using both SMW and EGN methods (Table 2). EGN achieves up to 1.6× speed-up compared to SMW on larger models ($d \geq 10^5$), which translates into substantial training-time savings since the solver is invoked at every iteration.

For completeness, we note here that another option to solve (14) exactly is through the QR factorization of $\mathbf{J}_t^\top$ (see pseudocode

Algorithm 1 EGN direction function.

1: **Input:** (pseudo-)residuals $\mathbf{r}$, stacked Jacobians $\mathbf{J}$, regularizer λ, batch size b.
2: Solve the linear system for δ: $\left(\mathbf{Q}\mathbf{J}\mathbf{J}^\top + b\lambda\mathbf{I}_{bc}\right)\delta = \mathbf{r}$ // $\mathcal{O}\left(b^2c^2d + b^3c^3\right)$
3: Calculate direction $\mathbf{d}^{\text{LM}} = -\mathbf{J}^\top\delta$ // $\mathcal{O}(bcd)$
4: **Return** $\mathbf{d}^{\text{LM}}$

Table 2
Wall-clock time (seconds, mean over 1000 runs) to solve (14) with SMW and EGN across different model sizes for $b = 32$, $c = 10$ on NVIDIA RTX A4000 GPU.

d	1K	10K	100K	1M	2M
SMW	0.0010	0.0013	0.0044	0.0368	0.0747
EGN	0.0008	0.0011	0.0028	0.0246	0.0517

in Appendix B). The complexity of such an approach is also $\mathcal{O}\left(b^2\, c^2\, d\right)$, dominated by the economy size QR decomposition of $\mathbf{J}^\top$. However, performing a QR decomposition is significantly more expensive than performing matrix–matrix multiplications and, in practice, EGN is preferred.

The CG method is an essential part of, e.g., HFO [32,34], SGN [27], Newton-CG [20], LM [43], and Distributed Newton's method with optimal shrinkage [59]. The complexity of CG approximately solving Eq. (14) is $\mathcal{O}(ld^2)$ where l is the number of CG iterations [56]. A typical number of CG iterations ranges from 3 in [34] to 50 in [60]. Note that, unless the number of classes c is high (which is one of the limitations of EGN addressed in Section 6.3), we have $ld^2 > b^2\, c^2\, d$, such that EGN solves the system both exactly and faster than CG.

4.2. Additional improvements

In addition to estimating the Hessian-adjusted direction, most state-of-the-art SOMs employ strategies to reduce variance of gradient and Hessian estimates, safeguard against exploding gradients, and dynamically adjust hyper-parameters to training steps. Next, we explore several enhancements to EGN, including momentum acceleration [9,21,34] to mitigate issue (d)–that is, the noise in Hessian estimates; line search [43, 47,61] to ensure steps are sufficiently short to decrease the loss; and adaptive regularization [23,34,43] to achieve faster convergence and simplify hyper-parameter tuning.

Momentum. A significant challenge in second-order methods is the noise introduced in Hessian estimates due to stochastic sampling. Consider the update direction $\mathbf{d}_t = -\mathbf{H}_t^{-1}\mathbf{g}_t$, where both the approximate Hessian $\mathbf{H}_t$ and the gradient $\mathbf{g}_t$ are stochastic estimates of the true derivatives of the empirical risk (2). Conditioning the gradient with a noisy Hessian inverse can lead to inaccurate descent directions, impeding convergence. To mitigate this issue, temporal averaging techniques (or *momentum*) are employed to stabilize updates and accelerate convergence by combining information from previous iterations with current estimates. In ML applications, common momentum variants include simple accumulation [6], exponential moving average (EMA) [7], bias-corrected EMA [9,21] and momentum with an extrapolation step [2].

For diagonal scaling methods, including first-order accelerated algorithms [6,7,9] and SOMs [21,22], the accumulated estimates of both first and second moments are kept separately resulting in $\mathcal{O}(d)$ space complexity. For Gauss-Newton methods we could alternatively reduce the variance of the Jacobian $\mathbf{J}$, e.g., with SVRG [62], SAGA [63] or SARAH [64], which results in a space complexity of $\mathcal{O}(bcd)$. Another option is to apply momentum to the descent direction $\mathbf{d}_t$, explored in [34,61], which requires storing a vector of size d. Since we never explicitly materialize either the Hessian or the gradient (Algorithm 1), we follow the latter approach with bias-corrected EMA by default.

Line search. Line search is a widely used technique in deterministic optimization [56] that iteratively adjusts the learning rate to satisfy some minimum criteria (e.g., Wolfe's conditions), ensuring adequate decrease in the loss function. Allowing α to automatically adapt to each training step can significantly reduce the need for manual tuning, which is often time-consuming and computationally expensive. However, extending line search methods to the stochastic setting poses challenges due to the inherent noise in gradient estimates, making it difficult to guarantee the same theoretical properties as in the deterministic case [47]. Despite these challenges, line search has been successfully applied in practice to both stochastic FOMs [61,65] and SOMs [36,43]. In EGN we adopt the strategy proposed by Ref. [61], which incorporates a reset mechanism at the beginning of each search to minimize the computational overhead of evaluating the loss function (Algorithm 4 in the Appendix).

Adaptive regularization. Adaptive regularization techniques for stochastic SOMs are explored in [23,32,34,43] modifying the original Levenberg-Marquardt rule to the stochastic setting. The central idea is to track ρ, defined as the ratio between the decrease in the actual loss function (3) and the decrease in the quadratic model $\mathcal{M}(\Delta\mathbf{w}) = \mathcal{L}_b\left(\mathbf{y}_t, \Phi(\mathbf{x}_t;\mathbf{w}_t)\right) + \mathbf{g}_t^\top\Delta\mathbf{w} + \frac{1}{2\,b}\Delta\mathbf{w}^\top\mathbf{J}_t^\top\mathbf{Q}_t\mathbf{J}_t\Delta\mathbf{w}$, where $\Delta\mathbf{w} = \mathbf{w} - \mathbf{w}_t$. This yields

$$\rho := \frac{\mathcal{L}_b\left(\mathbf{y}_t, \Phi(\mathbf{x}_t;\mathbf{w}_{t+1})\right) - \mathcal{L}_b\left(\mathbf{y}_t, \Phi(\mathbf{x}_t;\mathbf{w}_t)\right)}{\mathbf{g}_t^\top\Delta\mathbf{w} + \frac{1}{2\,b}\Delta\mathbf{w}^\top\mathbf{J}_t^\top\mathbf{Q}_t\mathbf{J}_t\Delta\mathbf{w}}, \tag{23}$$

which measures the accuracy of the quadratic model. In case ρ is small or negative, $\mathcal{M}(\Delta\mathbf{w})$ provides inaccurate approximation and the value λ is increased. Conversely, if ρ is large, λ is decreased to give more weight to $\mathcal{M}(\Delta\mathbf{w})$ (see Algorithm 5 in the Appendix). As empirically found by Ref. [34], compared to the deterministic LM the increase/decrease coefficients need to be less aggressive to reduce the oscillations of λ_t.

Algorithm 2 incorporates all the improvements discussed above, effectively addressing issues (a)–(d) associated with SOMs and eliminating the need for manual hyper-parameter tuning.

Algorithm 2 EGN.

Input: training dataset $\mathcal{D}$, initial weights $\mathbf{w}_0$, initial regularizer λ_0, momentum strength β.
Initialize momentum: $\mathbf{m}_0 = 0$
for t in $1...T$ **do**
  Sample a mini-batch $\mathcal{B}_t$ from $\mathcal{D}$
  Estimate $\mathbf{r}_t$ and $\mathbf{J}_t$ (e.g., via backpropagation)
  Find direction $\mathbf{d}_t$ via Algorithm 1
  Calculate the momentum term: $\mathbf{m}_t \leftarrow \beta\mathbf{m}_{t-1} + (1-\beta)\mathbf{d}_t$
  Update direction: $\mathbf{d}_t \leftarrow \frac{\mathbf{m}_t}{1-\beta^t}$
  Line search for α_t via Algorithm 4
  Update weights: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \alpha_t\mathbf{d}_t$
  Update λ_{t+1} via Algorithm 5
end for
Return $\mathbf{w}_t$

5. Convergence analysis

In this section, we analyze the convergence of EGN in the general non-convex setting.

In EGN, we consider the sequence of iterates $\{\mathbf{w}_t\}_{t\geq 1}$ where each $\mathbf{w}_t$ is computed via (4) with $\mathbf{d}_t \equiv \mathbf{d}_t^{\mathrm{LM}}$. We aim to minimize the function $\mathcal{L}_N(\mathbf{w})$ using, for each realization $\xi \sim P_\xi$, the Hessian estimator $\mathbf{H}(\mathbf{w},\xi)$ and the gradient estimator $\mathbf{g}(\mathbf{w},\xi)$. Then, at each iteration t, the mini-batch estimates $\mathbf{g}_t$ and $\mathbf{H}_t$ of the gradient and Hessian are

$$\mathbf{g}_t = \frac{1}{b}\sum_{\xi_i\in\mathcal{B}_t}\mathbf{g}(\mathbf{w},\xi_i) := \frac{1}{b}\mathbf{J}_t^\top\mathbf{r}_t, \tag{24}$$

$$\mathbf{H}_t = \frac{1}{b}\sum_{\xi_i\in\mathcal{B}_t}\mathbf{H}(\mathbf{w},\xi_i) := \frac{1}{b}\mathbf{J}_t^\top\mathbf{Q}_t\mathbf{J}_t. \tag{25}$$

In our analysis, we make use of the following assumptions.

Assumption 5.1. *The function $\mathcal{L}_N$ is lower-bounded on its domain, i.e., $-\infty < \mathcal{L}_N^* := \inf_{\mathbf{w}\in\mathbb{R}^d}\mathcal{L}_N(\mathbf{w})$. In addition, $\mathcal{L}_N$ is twice differentiable with Lipschitz continuous first-order derivatives, i.e., $\exists L_1 \in \mathbb{R}$ such that*

$$\left\|\nabla\mathcal{L}_N(\bar{\mathbf{w}}) - \nabla\mathcal{L}_N(\tilde{\mathbf{w}})\right\| \leq L_1\left\|\bar{\mathbf{w}} - \tilde{\mathbf{w}}\right\|, \quad \forall\bar{\mathbf{w}},\tilde{\mathbf{w}}\in\mathbb{R}^d. \tag{26}$$

Assumption 5.2. *At any iteration t, $\mathbf{g}(\mathbf{w}_t,\xi)$ is an unbiased estimator of $\nabla\mathcal{L}_N(\mathbf{w}_t)$, i.e.,*

$$\mathbb{E}_\xi[\mathbf{g}(\mathbf{w}_t,\xi)] = \nabla\mathcal{L}_N(\mathbf{w}_t). \tag{27}$$

Moreover, we have

$$\mathbb{E}_\xi\left[\left\|\mathbf{g}(\mathbf{w}_t,\xi) - \nabla\mathcal{L}_N(\mathbf{w}_t)\right\|^2\right] \leq \sigma_g^2, \tag{28}$$

where $\sigma_g > 0$ is a variance parameter.

Assumption 5.3. *At any iteration t, $\underline{\kappa}\mathbf{I} \leq \mathbf{Q}_t \leq \bar{\kappa}\mathbf{I}$ with $\bar{\kappa} \geq \underline{\kappa} \geq 0$. Additionally, $\exists \bar{\sigma}, \underline{\sigma}$ satisfying $0 < \underline{\sigma} \leq \bar{\sigma}$ such that $\underline{\sigma} \leq \|\mathbf{J}_t\| \leq \bar{\sigma}$ for all $t \geq 0$.*

Assumption 5.4. *At any iteration t, and for any random matrix $\mathbf{B}_t$ satisfying $\mathbf{B}_t \succeq \mu\mathbf{I}$ with $\mu > 0$, $\mathbb{E}\left[\langle \nabla \mathcal{L}_N(\mathbf{w}_t), \mathbf{B}_t \mathbf{g}_t \rangle | \mathbf{w}_t\right] \geq \mu K \|\nabla \mathcal{L}_N(\mathbf{w}_t)\| \|\mathbb{E}[\mathbf{g}_t | \mathbf{w}_t]\|$, where $K = (3(b\lambda_t + \bar{\kappa}\bar{\sigma}^2))/(5\, b\lambda_t)$.*

These assumptions are standard in stochastic optimization literature [41,66,67], and they hold naturally or can be easily enforced for typical neural network architectures and loss functions used in machine learning practice. Notably, the variance of the stochastic gradient estimator (Assumption 5.2) is more directly controllable from a practitioner's perspective by selecting a sufficiently large batch size or adding a momentum term that uses past gradients to inform the direction of update (see, e.g., [41,67,68]). We also remark that the matrix $\mathbf{B}_t$ in Assumption 5.4 may be interpreted as a preconditioner. Depending on the nature of its conditional correlation with $\mathbf{g}_t$ given $\mathbf{w}_t$, a large batch size can enhance the practicality of the assumption. In practice, either this conditional correlation is nonexistent (see the comment on [41, Assumption 4.3(b)]), or the contribution of $\mathbf{g}_t$ is inversely scaled by the (large) batch size as in the proof of Lemma 5.5 below.

In Lemma 5.5, we prove a descent lemma for EGN under the given conditions.

Lemma 5.5. *Let $\{\mathbf{w}_t\}$ be the sequence of iterates generated by (4) with $\mathbf{d}_t \equiv \mathbf{d}_t^{LM}$ and let Assumptions 5.1–5.3 hold. Suppose there exists $\alpha^{max} > 0$ such that $\alpha_t \leq \alpha^{max}$ for all t in Algorithm 4. Then,*

$$\mathbb{E}[\mathcal{L}_N(\mathbf{w}_{t+1})|\mathbf{w}_t] \leq \mathcal{L}_N(\mathbf{w}_t) - \frac{\alpha_t}{2}\|\nabla \mathcal{L}_N(\mathbf{w}_t)\|^2 + \frac{3\sigma_g^2 \alpha_t^2}{10\, b\lambda_t \alpha^{max}}. \tag{29}$$

Proof. From Assumption 5.1, we have

$$\begin{aligned}
\mathcal{L}_N(\mathbf{w}_{t+1}) &\leq \mathcal{L}_N(\mathbf{w}_t) + \langle \nabla \mathcal{L}_N(\mathbf{w}_t), \mathbf{w}_{t+1} - \mathbf{w}_t \rangle + \frac{L_1}{2}\left\|\mathbf{w}_{t+1} - \mathbf{w}_t\right\|^2 \\
&= \mathcal{L}_N(\mathbf{w}_t) - \alpha_t \left\langle \nabla \mathcal{L}_N(\mathbf{w}_t), \left(\frac{1}{b}\mathbf{J}_t^\top \mathbf{Q}_t \mathbf{J}_t + \lambda_t \mathbf{I}\right)^{-1} \mathbf{g}_t \right\rangle \\
&\quad + \frac{\alpha_t^2 L_1}{2}\left\| \left(\frac{1}{b}\mathbf{J}_t^\top \mathbf{Q}_t \mathbf{J}_t + \lambda_t \mathbf{I}\right)^{-1} \mathbf{g}_t \right\|^2.
\end{aligned} \tag{30}$$

Next, we set $\mathbf{B}_t = \left(\frac{1}{b}\mathbf{J}_t^\top \mathbf{Q}_t \mathbf{J}_t + \lambda_t \mathbf{I}\right)^{-1}$ in Assumption 5.4. From Assumption 5.3, we can show that $\mathbf{B}_t \succeq \mu\mathbf{I}$ with $\mu := b/(b\lambda_t + \bar{\kappa}\bar{\sigma}^2)$. Using this result and Assumption 5.3 in (30), we obtain

$$\mathcal{L}_N(\mathbf{w}_{t+1}) \leq \mathcal{L}_N(\mathbf{w}_t) - \alpha_t \langle \nabla \mathcal{L}_N(\mathbf{w}_t), \mathbf{B}_t \mathbf{g}_t \rangle + \frac{b^2 \alpha_t^2 L_1}{2(b\lambda_t + \underline{\kappa}\underline{\sigma}^2)^2}\|\mathbf{g}_t\|^2. \tag{31}$$

Notice that by Assumption 5.2, we have $\mathbb{E}[\mathbf{g}_t|\mathbf{w}_t] = \nabla \mathcal{L}_N(\mathbf{w}_t)$. Hence, the inequality in Assumption 5.4 can be written as $\mathbb{E}\left[\langle \nabla \mathcal{L}_N(\mathbf{w}_t), \mathbf{B}_t \mathbf{g}_t \rangle | \mathbf{w}_t\right] \geq \mu K \|\nabla \mathcal{L}_N(\mathbf{w}_t)\|^2$. Taking conditional expectation on both sides of (31) with respect to ξ and using Assumption 5.4, we get

$$\mathbb{E}[\mathcal{L}_N(\mathbf{w}_{t+1})|\mathbf{w}_t] \leq \mathcal{L}_N(\mathbf{w}_t) - \frac{3\alpha_t}{5\lambda_t}\|\nabla \mathcal{L}_N(\mathbf{w}_t)\|^2 + \frac{b^2 \alpha_t^2 L_1}{2(b\lambda_t + \underline{\kappa}\underline{\sigma}^2)^2}\mathbb{E}[\|\mathbf{g}_t\|^2|\mathbf{w}_t]. \tag{32}$$

Using Assumption 5.2, we have

$$\begin{aligned}
\mathbb{E}[\|\mathbf{g}_t\|^2|\mathbf{w}_t] &= \mathbb{E}[\|\mathbf{g}_t - \nabla \mathcal{L}_N(\mathbf{w}_t) + \nabla \mathcal{L}_N(\mathbf{w}_t)\|^2|\mathbf{w}_t] \\
&= \mathbb{E}[\|\nabla \mathcal{L}_N(\mathbf{w}_t)\|^2|\mathbf{w}_t] + 2\mathbb{E}[\langle \mathbf{g}_t - \nabla \mathcal{L}_N(\mathbf{w}_t), \nabla \mathcal{L}_N(\mathbf{w}_t)\rangle|\mathbf{w}_t] \\
&\quad + \mathbb{E}[\|\mathbf{g}_t - \nabla \mathcal{L}_N(\mathbf{w}_t)\|^2|\mathbf{w}_t] \\
&= \|\nabla \mathcal{L}_N(\mathbf{w}_t)\|^2 + \mathbb{E}[\|\mathbf{g}_t - \nabla \mathcal{L}_N(\mathbf{w}_t)\|^2|\mathbf{w}_t] \\
&\leq \|\nabla \mathcal{L}_N(\mathbf{w}_t)\|^2 + \frac{\sigma_g^2}{b}.
\end{aligned} \tag{33}$$

Now, using (33) in (32), we obtain

$$\mathbb{E}[\mathcal{L}_N(\mathbf{w}_{t+1})|\mathbf{w}_t] \leq \mathcal{L}_N(\mathbf{w}_t) - \left(\frac{3\alpha_t}{5\lambda_t} - \frac{b^2\alpha_t^2 L_1}{2(b\lambda_t + \underline{\kappa}\underline{\sigma}^2)^2}\right)\|\nabla \mathcal{L}_N(\mathbf{w}_t)\|^2 + \frac{b\sigma_g^2 \alpha_t^2 L_1}{2(b\lambda_t + \underline{\kappa}\underline{\sigma}^2)^2}. \tag{34}$$

By choosing α_t to be sufficiently small, we can set $\alpha^{\max} = 1/c_1$ in Algorithm 4, where $c_1 = (5\, b^2\lambda_t L_1)/(3(b\lambda_t + \underline{\kappa}\underline{\sigma}^2)^2$. With this selection, inequality (34) holds, which completes the proof. □

Assumption 5.4 is necessary to prove the result in Lemma 5.5 due to the correlation between $\mathbf{B}_t$ and $\mathbf{g}_t$. If these quantities were not correlated, then one could exploit Assumptions 5.2–5.3: by taking the expected value and using (27), one could then bound $\mathbf{B}_t \succeq \mu\mathbf{I}$ to obtain the desired bound with $K = 1$. Clearly, that would yield different constants in our next results, but their nature would remain unaltered. Note that this suggests alternative versions of EGN, in which the correlation is removed by construction at the expense of an increased computational burden. A thorough investigation of such schemes is beyond the scope of this paper.

Unlike classical stochastic quasi-Newton and SGD ($\mathbf{H}_t = \mathbf{I}_d$) methods, the loss decrease condition of EGN has an explicit nonlinear dependence on the batch size b. In the regime $b \gg 1$, the influence of the variance parameter σ_g diminishes.

We prove next the following result about the convergence of EGN.

Theorem 5.6. *Let the assumptions in Lemma 5.5 hold, and assume $\lambda_t \equiv \lambda$ is fixed for all t. Let $\alpha_t = \frac{\alpha_0}{(t+1)^a}$ for some $0 < \alpha_0 < 1$, $\frac{1}{2} < a < 1$. Then, the loss gradient approaches 0 in expectation as the iteration count $T \to \infty$.*

Proof. Following the proof of Lemma 5.5, we fix $\alpha^{\max} = 1/c_1$ in Algorithm 4, with $c_1 = (5\, b^2\lambda_t L_1)/(3(b\lambda_t + \underline{\kappa}\underline{\sigma}^2)^2$, and take the conditional expectation on both sides of (29). This yields

$$\mathbb{E}[\mathcal{L}_N(\mathbf{w}_{t+1})] \leq \mathbb{E}[\mathcal{L}_N(\mathbf{w}_t)] - \frac{1}{2}\alpha_t \mathbb{E}[\|\nabla \mathcal{L}_N(\mathbf{w}_t)\|^2] + C\alpha_t^2, \tag{35}$$

where $C := \frac{3\sigma_g^2 c_1}{10\, b\lambda_t}$. Summing (35) over $t = 0, 1, \ldots, T-1$, we obtain

$$\mathbb{E}[\mathcal{L}_N(\mathbf{w}_T)] \leq \mathbb{E}[\mathcal{L}_N(\mathbf{w}_0)] - \frac{1}{2}\sum_{t=0}^{T-1} \alpha_t \mathbb{E}[\|\nabla \mathcal{L}_N(\mathbf{w}_t)\|^2] + C\sum_{t=0}^{T-1}\alpha_t^2. \tag{36}$$

Assuming that the learning rates are given by $\alpha_t = \frac{\alpha_0}{(t+1)^a}$ for some $0 < \alpha_0 < 1$, $\frac{1}{2} < a < 1$, we have

$$\sum_{t=0}^{T-1}\alpha_t^2 = D < \infty.$$

Next, consider a random variable $\mathbf{z}_T$ satisfying $P[\mathbf{z}_T = \mathbf{w}_t] = \frac{1}{T}$, for all t. Consequently,

$$\mathbb{E}[\|\nabla \mathcal{L}_N(\mathbf{z}_T)\|^2] = \frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}[\|\nabla \mathcal{L}_N(\mathbf{w}_t)\|^2],$$

such that, using also $\alpha_{t_2} \leq \alpha_{t_1}$ for all $t_2 \geq t_1$, (36) becomes

$$\mathbb{E}[\mathcal{L}_N(\mathbf{w}_T)] \leq \mathbb{E}[\mathcal{L}_N(\mathbf{w}_0)] - \alpha_{T-1}\frac{1}{2}T\mathbb{E}[\|\nabla \mathcal{L}_N(\mathbf{z}_T)\|^2] + CD.$$

Then, using $\mathbb{E}[\mathcal{L}_N(\mathbf{w}_T)] \geq \mathcal{L}_N^\star$ we have

$$\begin{aligned}
\mathbb{E}[\|\nabla \mathcal{L}_N(\mathbf{z}_T)\|^2] &\leq 2\frac{\mathbb{E}[\mathcal{L}_N(\mathbf{w}_0)] - \mathcal{L}_N^*}{\alpha_{T-1}T} + \frac{2CD}{\alpha_{T-1}T} \\
&= 2\frac{\mathbb{E}[\mathcal{L}_N(\mathbf{w}_0)] - \mathcal{L}_N^*}{\alpha_0\, T^{1-a}} + \frac{2CD}{\alpha_0\, T^{1-a}}.
\end{aligned}$$

Taking the limit for $T \to \infty$ we finally have

$$\lim_{T\to\infty}\mathbb{E}[\|\nabla \mathcal{L}_N(\mathbf{z}_T)\|^2] \leq 0.$$

□

6. Experiments

We conduct a series of experiments to measure the performance of EGN on several supervised learning and reinforcement learning tasks.

We select five baseline solvers: SGD [1], Adam [9], SGD with momentum and Gradient Activation Function (GAF) [69], a Quasi-Newton solver (SQN) [24,25,41], and SGN [27]. SGD acts as a most basic baseline with the computationally cheapest update; Adam is a widely used accelerated FOM for training DNNs; GAF is a recent first-order variant that reports faster convergence on deep networks; SGN is an *inexact* Gauss-Newton solver against which we evaluate the practical advantages of solving the system (5) *exactly*; and SQN acts as an alternative to Gauss-Newton that approximates the Hessian via low-rank updates. Since first-order methods typically require fewer computations per iterate, in order to obtain a fair comparison we monitor the wall time instead of the number of iterations. The learning rates are selected as the best performing α after a grid search in the logspace $\alpha \in [10^{-9}, 1]$. Additionally, for SGN we search for the optimal "number of CG iterations" within the set $\{3, 5, 10, 20, 50\}$. For EGN we introduce two extra hyper-parameters: "line search" $\{\text{True}, \text{False}\}$ and "momentum" $\{0.0, 0.9\}$. The best performing sets of hyper-parameters as well as detailed description of the datasets are available in Appendix C. The size of the mini-batch for all problems is 128. All the experiments are conducted on the Tesla T4 GPU in the Google Colab environment with float32 precision.

6.1. Supervised learning

For the regression task, we select three datasets: California Housing [70], Superconductivity [71], and Diamonds [72] with 20640, 21263, and 53940 training samples, respectively. For classification, we use the IMDB Reviews dataset [73] containing 25000 instances of movie reviews. Across all problems, the model Φ is a Feedforward Neural Network (FFNN) with three dense layers of 32, 64 and 32 units followed by the ReLU activation function with a total of 4449 parameters ($d = 4449$). The loss function during the training is a least-squares loss (10) for regression and softmax cross-entropy loss (12) for classification. The datasets are split into training and test sets in the proportion of 90/10 percent. Numerical features are scaled and categorical features are one-hot encoded. To measure the performance, we plot the evolution of the evaluation metric on unseen data (test set) with respect to wall time (in seconds). The evaluation metric is Root Mean Squared Error (RMSE) for regression and accuracy for classification.

The results are presented in Fig. 1 and Table 3. On all but the IMDB Reviews dataset EGN has achieved both faster convergence and lower test set error than any other optimization algorithm. On IMDB Reviews SGN is faster than EGN, however, the two solvers achieve the same accuracy after reaching convergence.

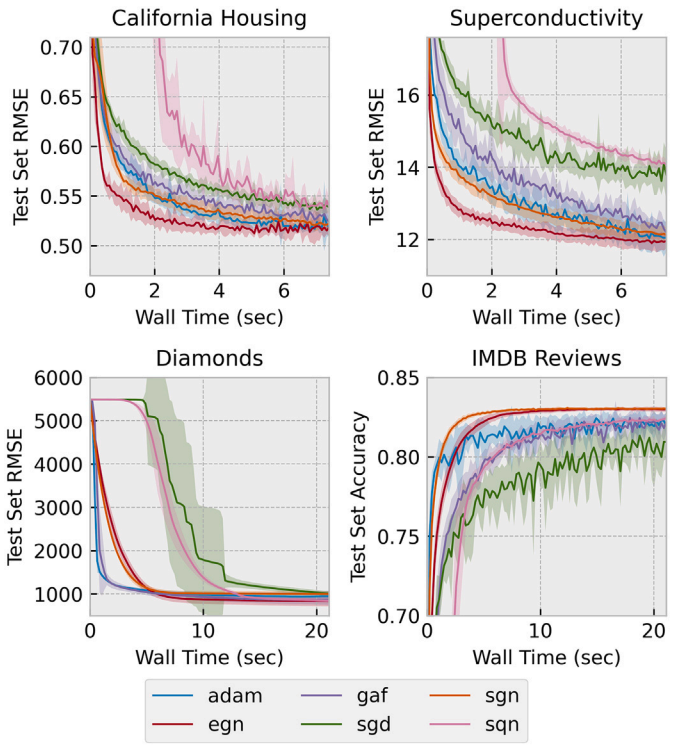


Fig. 1. Learning curves on the test set for SGD, Adam, GAF, SQN, SGN and EGN. The shaded area represents ±1 standard deviation around the mean (thick line) for 10 seeds.

6.2. Reinforcement learning

We demonstrate the application of EGN to reinforcement learning in two scenarios: continuous action spaces with Linear-Quadratic Regulator (LQR) and discrete action spaces using Deep Q-Network (DQN) [74].

Learning LQR controllers. Given a discrete time-invariant linear system with continuous states and actions, and a quadratic reward function our task is to learn the optimal value function $v^*(s)$ and the optimal policy $\pi^*(s)$ such that we maximize the cumulative return. Such problems can be solved in a data-driven fashion with the policy iteration procedure [75] (outlined in Appendix C.2). It is well known that the optimal value function is quadratic and the optimal policy function is linear [76]. Consequently, we define Φ as a quadratic function of states and actions. We track the norm of the difference between the optimal LQR controller calculated analytically knowing the system matrices and the learned weights of the model.

We select two linear systems from the Compleib set of benchmarks [77]. The first system is a deterministic model of a binary distillation tower (BDT) [78], and the second one represents the linearized vertical plane dynamics of an aircraft (UAV) with noise [79]. The results are displayed in the top two charts of Fig. 2 and Table 4. Both EGN and SGN outperform first-order methods by a considerable margin, with EGN enjoying slightly faster convergence in both cases, while SGN achieves a marginally lower error on the stochastic LQR upon reaching convergence.

Reinforcement learning with DQN. Adopting the problem formulation of Ref. [74], we aim to learn the weights of a neural network that represent a Q-value function $q(s, a)$ that maps states s and actions a into scores

Table 3
Performance after training completion (supervised learning).

Optimizer	California housing	Superconduct	Diamonds	IMDB reviews
SGD	0.539 ± 0.006	13.788 ± 0.698	1008.936 ± 54.940	0.809 ± 0.011
Adam	0.519 ± 0.009	12.052 ± 0.381	947.258 ± 130.079	0.820 ± 0.008
GAF	0.524 ± 0.006	12.193 ± 0.293	857.817 ± 109.033	0.822 ± 0.006
EGN	**0.518 ± 0.009**	**11.961 ± 0.207**	**840.500 ± 126.444**	**0.830 ± 0.001**
SGN	0.522 ± 0.007	12.121 ± 0.196	998.688 ± 61.489	**0.830 ± 0.001**
SQN	0.539 ± 0.008	14.070 ± 0.141	844.951 ± 48.109	0.824 ± 0.001

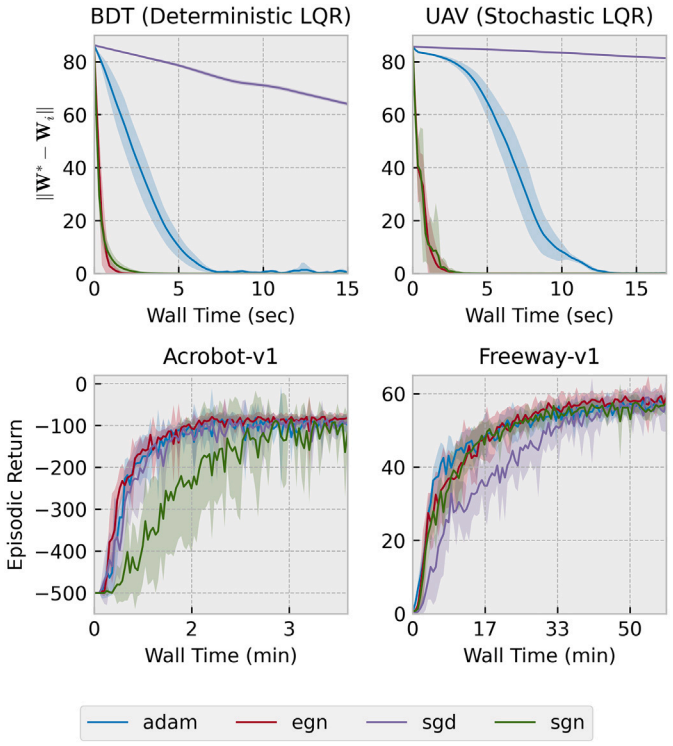


Fig. 2. Learning curves for SGD, Adam, SGN and EGN. The shaded area represents ±1 standard deviation around the mean return (thick line) for 10 seeds.

(Q-values) for a discrete set of actions. Once training is complete, the optimal policy is formed by calculating the Q-value of each action and choosing the highest-scoring action.

We build upon CleanRL [80] framework for running RL experiments, selecting two environments: Acrobot-v1 and Freeway-v1. Acrobot-v1 is an OpenAI gym [81] environment with a 6-dimensional state vector and a set of 3 discrete actions where the goal is to swing the free end of the connected joints above a given height in as few steps as possible. Freeway-v1 is a MinAtar [82] environment that emulates the original Freeway Atari game. The state is represented by a 10×10 image and there are 3 discrete actions available. For Acrobot-v1 the network Φ is a FFNN with three dense layers of 32, 64 and 32 units followed by the ReLU activation function with a total of 4515 parameters. For Freeway-v1 we design a compact CNN, comprising of a convolutional layer with sixteen 3×3 filters and ReLU activation, followed by flattening, a 64-unit dense layer with ReLU, and a final dense layer outputting Q-values for all actions ($d = 103683$).

The cumulative returns from each completed episode are recorded and displayed in the bottom two charts of Fig. 2. The results for both Acrobot-v1 and Freeway-v1 show no distinct advantage among the solvers, as they all reach similar episodic returns. We notice, however, that EGN slightly outperforms other optimizers by achieving a higher return level at convergence (see Table 4).

6.3. Limitations

Explicit gradients. Unlike first-order methods that rely on the average gradient of the batch loss, Gauss-Newton methods require the full Jacobian matrix, which contains the gradients of each sample. As a result, backpropagation for EGN is more time-consuming than for FOMs. Moreover, this can lead to increased GPU memory usage, especially with high-dimensional parameter vectors.

Large batch sizes. In our experiments we observed that the cost of computing the derivatives and the subsequent cost of computing the step (Algorithm 1) are comparable. However, for large batch sizes ($b > 128$) we observed that the computational times increased significantly. Fig. 3 quantifies how the direction calculation stage begins to dominate the update time as the batch size grows. While EGN remains an efficient drop-in replacement for first-order methods within the commonly-used range $32 \leq b \leq 128$, scaling to very large batches will require additional techniques. We suspect this to be related to hardware limitations and leave a systematic study of such mitigations to future work.

Large number of classes for multi-class classification. By incorporating the softmax function into the loss Eq. (12) we introduce coupling between the individual outputs of Φ in the denominator $\sum_{j=1}^{c} e^{\mathbf{z}_{i,j}}$, which makes the Jacobian $\mathbf{J}$ a dense $bc \times d$ matrix. Speeding up the calculation of $\mathbf{J}$ remains an open research question and constitutes a major obstacle in using Gauss-Newton methods for tasks involving a large number of classes, e.g., LLM pre-training. One possible solution could consist in moving the softmax function directly into the model Φ as the last layer of the network and then only computing the Jacobian with respect to the correct class, resulting in $\mathbf{J} \in \mathbb{R}^{b \times d}$. This approach, however, yields a Hessian approximation that becomes singular close to the solution,

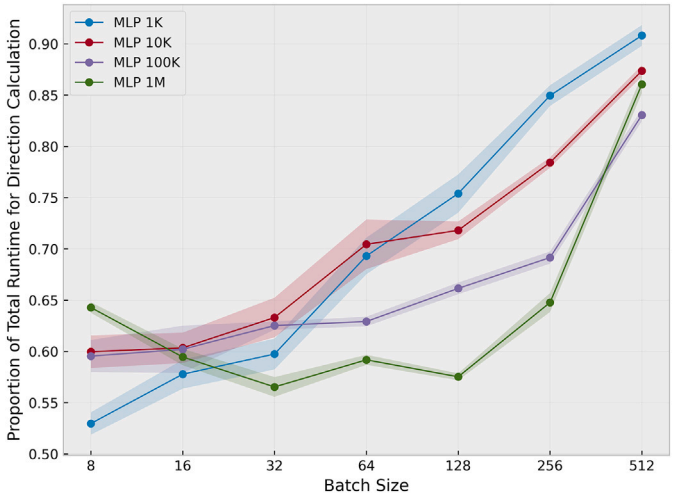


Fig. 3. Proportion of total update time spent in Algorithm 1 as a function of batch size for fully-connected neural networks (MLPs) of various sizes. Curves show the mean over 1000 runs; shaded regions denote ±1 standard deviation. Absolute wall-clock times are reported in Appendix C.4.

Table 4
Performance after training completion (reinforcement learning).

Optimizer	BDT	UAV	Acrobot-v1	Freeway-v1
SGD	64.058 ± 0.768	81.378 ± 0.306	−90.962 ± 13.484	56.996 ± 7.359
Adam	0.443 ± 0.297	0.157 ± 0.100	−98.489 ± 20.827	55.967 ± 2.251
EGN	**0.000 ± 0.000**	0.043 ± 0.031	**−81.796 ± 11.778**	**58.916 ± 1.946**
SGN	**0.000 ± 0.000**	**0.033 ± 0.019**	−114.052 ± 39.914	57.424 ± 3.611

which results in numerical instabilities that hinder convergence [83]. Another possibility suggested in [84] consists of replacing the softmax cross-entropy loss with the multi-class hinge loss. Although the computation becomes faster for the hinge loss, empirical evidence shows that the test set accuracy upon training completion is higher for the CE loss. Finally, a promising approach is the Gauss-Newton-Bartlett (GNB) estimator proposed by Ref. [22], which replaces the exact LM Hessian with an approximation obtained by sampling the subset of predicted labels.

7. Conclusion

We presented the EGN algorithm, a stochastic second-order method that efficiently estimates the descent direction by using a low-rank Gauss-Newton Hessian approximation and leveraging the Duncan-Guttman matrix identity. We demonstrated that EGN solves the system $\mathbf{H}^{\text{LM}}\mathbf{d} = -\mathbf{g}$ exactly with less computational burden than other exact Gauss-Newton methods, as well as inexact methods that rely on conjugate gradient iterates. We also proved that under mild assumptions our algorithm converges in expectation. Our empirical results show that EGN consistently matches or exceeds the generalization performance of well-tuned SGD, Adam, GAF, SQN, and SGN optimizers across various supervised and reinforcement learning tasks.

Future work will focus on addressing the shortcomings of EGN in classification problems with a large number of classes. A promising direction is to approximate the Gauss-Newton Hessian matrix to avoid computing the full Jacobian of the network, e.g., using techniques such as the Gauss-Newton-Bartlett estimator [22]. Another direction is to study the performance of EGN on larger datasets and more complex models.

CRediT authorship contribution statement

Mikalai Korbit: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Resources, Project administration, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Adeyemi D. Adeoye:** Writing – review & editing, Methodology, Formal analysis. **Alberto Bemporad:** Writing – review & editing, Project administration, Methodology. **Mario Zanon:** Writing – review & editing, Writing – original draft, Supervision, Project administration, Methodology, Formal analysis, Conceptualization.

Declaration of competing interest

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

Acknowledgments

This work was partially funded by the European Union (ERC Advanced Research Grant COMPACT, No. 101141351). Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.

Appendix A. Proofs and derivations

A.1. Generalized gauss-newton hessian approximation

Claim. The generalized Gauss-Newton Hessian approximation scheme for the batch loss (3) is $\mathbf{H}^{\text{GN}} = \frac{1}{b}\mathbf{J}^\top\mathbf{Q}\mathbf{J}$.

Proof. The derivative of the generic batch loss function (3) reads

$$\frac{\partial}{\partial \mathbf{w}}\mathcal{L}_b = \frac{\partial}{\partial \mathbf{w}}\left[\frac{1}{b}\sum_{i=1}^{b}\ell\left(\mathbf{y}_i, \Phi(\mathbf{x}_i;\mathbf{w})\right)\right]$$

Employing the chain rule $\frac{\partial \ell}{\partial \mathbf{w}} = \frac{\partial \ell}{\partial \Phi}\frac{\partial \Phi}{\partial \mathbf{w}}$ we have

$$\frac{\partial}{\partial \mathbf{w}}\mathcal{L}_b = \frac{1}{b}\sum_{i=1}^{b}\frac{\partial \ell\left(\mathbf{y}_i, \Phi(\mathbf{x}_i;\mathbf{w})\right)}{\partial \Phi}\frac{\partial \Phi(\mathbf{x}_i;\mathbf{w})}{\partial \mathbf{w}} = \frac{1}{b}\sum_{i=1}^{b}\mathbf{L}_i\mathbf{J}_{\Phi_i},$$

where $\mathbf{L}_i = \frac{\partial \ell(\mathbf{y}_i, \Phi(\mathbf{x}_i;\mathbf{w}))}{\partial \Phi} \in \mathbb{R}^{1\times c}$ and $\mathbf{J}_{\Phi_i} = \frac{\partial \Phi(\mathbf{x}_i;\mathbf{w})}{\partial \mathbf{w}} \in \mathbb{R}^{c\times d}$. We obtain the Hessian by differentiating the gradient

$$\mathbf{H} = \frac{\partial}{\partial \mathbf{w}}\left[\frac{1}{b}\sum_{i=1}^{b}\mathbf{L}_i\mathbf{J}_{\Phi_i}\right].$$

Using the product rule we arrive at

$$\mathbf{H} = \frac{1}{b}\sum_{i=1}^{b}\left(\frac{\partial}{\partial \mathbf{w}}\left[\mathbf{L}_i\right]\mathbf{J}_{\Phi_i} + \sum_{k=1}^{c}\mathbf{L}_{i,k}\frac{\partial}{\partial \mathbf{w}}\left[\mathbf{J}_{\Phi_{i,k}}\right]\right),$$

where $\mathbf{L}_{i,k} \in \mathbb{R}$ is the k-th element of the derivative of the loss with respect to Φ, and $\mathbf{J}_{\Phi_{i,k}} \in \mathbb{R}^{1\times d}$ is the k-th row the Jacobian $\mathbf{J}_{\Phi_i}$. We obtain the Gauss-Newton Hessian approximation by neglecting the second-term of $\mathbf{H}$ [17,53], such that

$$\mathbf{H}^{\text{GN}} = \frac{1}{b}\sum_{i=1}^{b}\frac{\partial}{\partial \mathbf{w}}\left[\mathbf{L}_i\right]\mathbf{J}_{\Phi_i}.$$

Given that

$$\frac{\partial}{\partial \mathbf{w}}\left[\mathbf{L}_i\right] = \mathbf{J}_{\Phi_i}^\top\frac{\partial^2 \ell\left(\mathbf{y}_i, \Phi(\mathbf{x}_i;\mathbf{w})\right)}{\partial \Phi^2} = \mathbf{J}_{\Phi_i}^\top\mathbf{Q}_{\ell_i},$$

where $\mathbf{Q}_{\ell_i} = \frac{\partial^2 \ell(\mathbf{y}_i, \Phi(\mathbf{x}_i;\mathbf{w}))}{\partial \Phi^2} \in \mathbb{R}^{c\times c}$ is the second derivative of the loss with respect to the function's output, we have

$$\mathbf{H}^{\text{GN}} = \frac{1}{b}\sum_{i=1}^{b}\mathbf{J}_{\Phi_i}^\top\mathbf{Q}_{\ell_i}\mathbf{J}_{\Phi_i}.$$

Or using the compact notation

$$\mathbf{H}^{\text{GN}} = \frac{1}{b}\mathbf{J}^\top\mathbf{Q}\mathbf{J},$$

where we vertically stack individual Jacobians $\mathbf{J}_{\Phi_i}$ for each sample in the batch $\mathcal{B}$ to form $\mathbf{J} \in \mathbb{R}^{bc\times d}$ and form a block diagonal matrix $\mathbf{Q} = \text{blkdiag}(\mathbf{Q}_{\ell_1}, \mathbf{Q}_{\ell_2}, \dots, \mathbf{Q}_{\ell_b}) \in \mathbb{R}^{bc\times bc}$. □

A.2. Gauss-newton hessian of the MSE loss function

Claim. The gradient of the batch MSE loss is $\mathbf{g} = \frac{1}{b}\mathbf{J}^\top\mathbf{r}$.

Proof. We define the residual vector $\mathbf{r}$ as $\mathbf{r} := \begin{bmatrix}\Phi(\mathbf{x}_1;\mathbf{w}) - \mathbf{y}_1 & \dots & \Phi(\mathbf{x}_b;\mathbf{w}) - \mathbf{y}_b\end{bmatrix}^\top$. Recall that the stacked Jacobians of the neural network are denoted by $\mathbf{J}$, with $\mathbf{J} \in \mathbb{R}^{b\times d}$ for the regression task. The batch loss (3) for MSE is

$$\mathcal{L}_b(\mathbf{w}) = \frac{1}{2b}\sum_{i=1}^{b}\left(\Phi(\mathbf{x}_i;\mathbf{w}) - \mathbf{y}_i\right)^2 = \frac{1}{2b}\mathbf{r}^\top\mathbf{r}.$$

So that

$$\frac{\partial}{\partial \mathbf{w}}\left[\mathcal{L}_b\right] = \frac{\partial \mathcal{L}_b}{\partial \mathbf{r}}\frac{\partial \mathbf{r}}{\partial \Phi}\frac{\partial \Phi}{\partial \mathbf{w}} = \frac{1}{b}\mathbf{r}^\top\mathbf{J}.$$

Since we define the gradient to be a column vector

$$\mathbf{g} = \left(\frac{\partial \mathcal{L}_b}{\partial \mathbf{w}}\right)^\top = \left(\frac{1}{b}\mathbf{r}^\top\mathbf{J}\right)^\top = \frac{1}{b}\mathbf{J}^\top\mathbf{r}.$$

□

Claim. The Hessian of the batch MSE loss is $\mathbf{H}^{\text{GN}} = \frac{1}{b}\mathbf{J}^\top\mathbf{J}$.

Proof. We obtain the Hessian by differentiating the gradient of the loss

$$\mathbf{H} = \frac{\partial}{\partial \mathbf{w}}\mathbf{g} = \frac{1}{b}\frac{\partial}{\partial \mathbf{w}}\left(\mathbf{J}^\top\mathbf{r}\right).$$

Using the product rule of vector calculus

$$\mathbf{H} = \frac{1}{b}\left(\mathbf{J}^\top\frac{\partial}{\partial \mathbf{w}}\left[\mathbf{r}\right] + \left(\frac{\partial}{\partial \mathbf{w}}\left[\mathbf{J}^\top\right]\right)\mathbf{r}\right) = \frac{1}{b}\left(\mathbf{J}^\top\mathbf{J} + \sum_{i=1}^{b}\frac{\partial^2}{\partial \mathbf{w}^2}\left[\Phi(\mathbf{x}_i;\mathbf{w})\right]\mathbf{r}_i\right).$$

where $\mathbf{r}_i \in \mathbb{R}$ is the i-th element of $\mathbf{r}$.

Neglecting the second term we obtain the Gauss-Newton approximation of the Hessian

$$\mathbf{H}^{\text{GN}} = \frac{1}{b}\mathbf{J}^\top\mathbf{J}.$$

□

A.3. Gauss-newton hessian of the multi-class cross-entropy loss function

Claim. The gradient of the batch multi-class cross-entropy loss is $\mathbf{g} = \frac{1}{b}\mathbf{J}^\top\mathbf{r}$.

Proof. We recall from Appendix A.1 that the partial derivative of the generic loss function is

$$\frac{\partial}{\partial \mathbf{w}}\mathcal{L}_b = \frac{1}{b}\sum_{i=1}^{b}\frac{\partial \ell\left(\mathbf{y}_i, \Phi(\mathbf{x}_i;\mathbf{w})\right)}{\partial \Phi}\mathbf{J}_{\Phi_i}.$$

Assuming that $\mathbf{J}_{\Phi_i}$ is calculated during the backpropagation stage, we examine the term $\frac{\partial \ell(\mathbf{y}_i, \Phi(\mathbf{x}_i;\mathbf{w}))}{\partial \Phi}$. The cross-entropy loss function (12) is defined as

$$\ell\left(\mathbf{y}_i, \Phi(\mathbf{x}_i;\mathbf{w})\right) := -\mathbf{y}_i^\top \log\left(\sigma\left(\Phi(\mathbf{x}_i;\mathbf{w})\right)\right),$$

where σ is the softmax function defined as

$$\sigma(\mathbf{z}_{i,k}) = \frac{e^{\mathbf{z}_{i,k}}}{\sum_{j=1}^{c} e^{\mathbf{z}_{i,j}}},$$

with a c-dimensional vector of prediction scores (logits) $\mathbf{z}_i = \Phi(\mathbf{x}_i;\mathbf{w})$ and a c-dimensional vector of probabilities $\mathbf{p}_i = \sigma(\mathbf{z}_i)$. Applying the chain rule and using the shorthand notation we obtain

$$\frac{\partial \ell\left(\mathbf{y}_i, \Phi(\mathbf{x}_i;\mathbf{w})\right)}{\partial \Phi} = \frac{\partial \ell}{\partial \mathbf{p}_i}\frac{\partial \mathbf{p}_i}{\partial \mathbf{z}_i}\frac{\partial \mathbf{z}_i}{\partial \Phi}.$$

Differentiating the loss with respect to the softmax function yields

$$\frac{\partial \ell}{\partial \mathbf{p}_i} = \left[-\frac{\mathbf{y}_{i,1}}{\mathbf{p}_{i,1}}, -\frac{\mathbf{y}_{i,2}}{\mathbf{p}_{i,2}}, \dots, -\frac{\mathbf{y}_{i,c}}{\mathbf{p}_{i,c}}\right].$$

The derivative of the k-th element of the softmax function wrt the l-th input $\mathbf{z}_{i,l}$ is split in two cases

$$\frac{\partial \mathbf{p}_{i,k}}{\partial \mathbf{z}_{i,l}} = \begin{cases} \mathbf{p}_{i,k}(1-\mathbf{p}_{i,k}), & \text{if } k = l \\ -\mathbf{p}_{i,k}\mathbf{p}_{i,l}, & \text{if } k \neq l. \end{cases}$$

So that the derivative $\frac{\partial \mathbf{p}_i}{\partial \mathbf{z}_i} \in \mathbb{R}^{c\times c}$ can be structured as

$$\frac{\partial \mathbf{p}_i}{\partial \mathbf{z}_i} = \text{diag}(\mathbf{p}_i) - \mathbf{p}_i\mathbf{p}_i^\top.$$

Combining $\frac{\partial \ell}{\partial \mathbf{p}_i}$ and $\frac{\partial \mathbf{p}_i}{\partial \mathbf{z}_i}$, we obtain

$$\frac{\partial \ell}{\partial \mathbf{z}_{i,k}} = -\sum_{l=1}^{c}\left(\frac{\mathbf{y}_{i,l}}{\mathbf{p}_{i,l}}\right)\begin{cases} \mathbf{p}_{i,k}(1-\mathbf{p}_{i,k}), & \text{if } k = l \\ -\mathbf{p}_{i,k}\mathbf{p}_{i,l}, & \text{if } k \neq l, \end{cases}$$

Which simplifies to

$$\frac{\partial \ell}{\partial \mathbf{z}_{i,k}} = -\mathbf{y}_{i,k} + \mathbf{p}_{i,k} = \mathbf{p}_{i,k} - \mathbf{y}_{i,k}.$$

Since $\frac{\partial \mathbf{z}_i}{\partial \Phi}$ is the identity, we have

$$\frac{\partial \ell\left(\mathbf{y}_i, \Phi(\mathbf{x}_i;\mathbf{w})\right)}{\partial \Phi} = \mathbf{p}_i - \mathbf{y}_i.$$

The same way as for the MSE loss, we define (pseudo-)residuals $\mathbf{r}_i \in \mathbb{R}^c$ as a column vector of probabilities minus the targets, i.e., $\left(\mathbf{p}_i - \mathbf{y}_i\right)^\top$. Substituting back into the loss derivative

$$\frac{\partial \mathcal{L}_b}{\partial \mathbf{w}} = \frac{1}{b}\sum_{i=1}^{b}\mathbf{r}_i^\top\mathbf{J}_{\Phi_i}.$$

Since we define the gradient to be a column vector

$$\mathbf{g} = \left(\frac{\partial \mathcal{L}_b}{\partial \mathbf{w}}\right)^\top = \left(\frac{1}{b}\sum_{i=1}^{b}\mathbf{r}_i^\top\mathbf{J}_{\Phi_i}\right)^\top = \frac{1}{b}\sum_{i=1}^{b}\mathbf{J}_{\Phi_i}^\top\mathbf{r}_i.$$

Or in shorthand notation

$$\mathbf{g} = \frac{1}{b}\mathbf{J}^\top\mathbf{r}$$

where for each sample i in the batch $\mathcal{B}$ we vertically stack Jacobians $\mathbf{J}_{\Phi_i}$ as well as residuals $\mathbf{r}_i$ to form $\mathbf{J} \in \mathbb{R}^{bc\times d}$ and $\mathbf{r} \in \mathbb{R}^{bc}$. □

Claim. The Hessian of the batch multi-class cross-entropy loss is $\mathbf{H}^{\text{GN}} = \frac{1}{b}\mathbf{J}^\top\mathbf{Q}\mathbf{J}$.

Proof. Recall that the GN Hessian for the generalized case (Appendix A.1) is

$$\mathbf{H}^{\text{GN}} = \frac{1}{b}\sum_{i=1}^{b}\mathbf{J}_{\Phi_i}^\top\mathbf{Q}_{\ell_i}\mathbf{J}_{\Phi_i}. \tag{A.1}$$

The second derivative of the loss with respect to the function's output $\mathbf{Q}_{\ell_i} = \frac{\partial^2 \ell(\mathbf{y}_i, \Phi(\mathbf{x}_i;\mathbf{w}))}{\partial \Phi^2} \in \mathbb{R}^{c\times c}$ for CE loss is

$$\mathbf{Q}_{\ell_i} = \frac{\partial}{\partial \Phi}\left[\mathbf{p}_i - \mathbf{y}_i\right].$$

Using the gradient of the softmax function derived earlier, we form a symmetric matrix $\mathbf{Q}_{\ell_i}$ such that

$$\mathbf{Q}_{\ell_i} = \begin{bmatrix} \mathbf{p}_{i1}(1-\mathbf{p}_{i1}) & -\mathbf{p}_{i1}\mathbf{p}_{i2} & \cdots & -\mathbf{p}_{i1}\mathbf{p}_{ic} \\ -\mathbf{p}_{i2}\mathbf{p}_{i1} & \mathbf{p}_{i2}(1-\mathbf{p}_{i2}) & \cdots & -\mathbf{p}_{i2}\mathbf{p}_{ic} \\ \vdots & \vdots & \ddots & \vdots \\ -\mathbf{p}_{ic}\mathbf{p}_{i1} & -\mathbf{p}_{ic}\mathbf{p}_{i2} & \cdots & \mathbf{p}_{ic}(1-\mathbf{p}_{ic}) \end{bmatrix},$$

Or in compact notation

$$\mathbf{H}^{\text{GN}} = \frac{1}{b}\mathbf{J}^\top\mathbf{Q}\mathbf{J},$$

Where $\mathbf{Q}$ is a block diagonal matrix

$$\mathbf{Q} = \begin{bmatrix} \mathbf{Q}_{\ell_1} & 0 & 0 & 0 \\ 0 & \mathbf{Q}_{\ell_2} & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & \mathbf{Q}_{\ell_b} \end{bmatrix}$$

And $\mathbf{J} \in \mathbb{R}^{bc\times d}$ is vertically stacked Jacobians $\mathbf{J}_{\Phi_i}$.

□

Appendix B. Algorithms

Algorithm 3 Calculate direction using QR factorization (MSE loss).

Input: (pseudo-)residuals $\mathbf{r}$, stacked Jacobians of the model $\mathbf{J}$, regularizer λ.
Factorize $\mathbf{J}^\top$ with economy sized QR: $\mathbf{Q}, \mathbf{R} \leftarrow \text{qr}\left(\mathbf{J}^\top\right)$
Factorize: $\tilde{\mathbf{Q}}, \tilde{\mathbf{R}} \leftarrow \text{qr}\left(\mathbf{R}\mathbf{R}^\top + \lambda \mathbf{I}\right)$
Solve the linear system for δ: $\tilde{\mathbf{R}}\delta = \tilde{\mathbf{Q}}^\top \mathbf{R}\mathbf{r}$
Calculate $\mathbf{d}^{\text{LM}} = -\mathbf{Q}\delta$
Return $\mathbf{d}^{\text{LM}}$

Algorithm 4 Armijo line search.

Input: direction $\mathbf{d}_t$, hyper-parameters $\alpha^{\max}$,κ, c^{up}, c^{down}.
Initialize $\alpha_t \leftarrow \min\left\{\alpha^{\max}, \alpha_{t-1}c^{\text{up}}\right\}$
while $\mathcal{L}\left(\mathbf{w}_{t+1}\right) > \mathcal{L}\left(\mathbf{w}_t\right) + \kappa\alpha_t \nabla\mathcal{L}\left(\mathbf{w}_t\right)^\top \mathbf{d}_t$ **do**
    Update $\alpha_t \leftarrow \alpha_t c^{\text{down}}$
    Update $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \alpha_t \mathbf{d}_t$
end while
Return α_t

Algorithm 5 Adaptive regularization [34].

Input: batch $\mathcal{B}_t$, current weights $\mathbf{w}_t$, updated weights $\mathbf{w}_{t+1}$.
Calculate ρ according to (23)
if $\rho < 0.25$ **then**
    $\lambda_{t+1} \leftarrow 1.01\lambda_t$
else if $\rho > 0.75$ **then**
    $\lambda_{t+1} \leftarrow 0.99\lambda_t$
else
    $\lambda_{t+1} \leftarrow \lambda_t$
end if
Return λ_{t+1}

Appendix C. Experiment details

C.1. Supervised learning

California Housing [70], a part of the scikit-learn [85] datasets package, consists of 20640 samples with 8 numerical features $\mathbf{x}$ encoding relevant information, e.g., location, median income, etc.; and a real-valued target $\mathbf{y}$ representing the median house value in California as recorded by the 1990 U.S. Census.

Superconductivity [71] is a dataset of HuggingFace Datasets [86] that contains 21263 instances of 79 numerical attributes (features $\mathbf{x}$) and critical temperatures (target $\mathbf{y}$) of superconductors.

Diamonds [72] is a TFDS [87] dataset containing 53940 instances of 9 physical attributes (both numerical and categorical features $\mathbf{x}$) and prices (target $\mathbf{y}$) of diamonds.

IMDB Reviews [73] is a TFDS [87] dataset that contains 25000 training samples and 25000 testing samples of movie reviews in a text format. Before passing the samples to the model Φ, we pre-process the raw text data with spaCy [88] *en_core_web_lg* pipeline which converts a text review into a 300-dimensional vector of numbers.

The optimal sets of hyper-parameters are presented in Table C.5.

C.2. Learning LQR controllers

Here, we define the problem of learning an LQR controller more formally. Given a discrete time-invariant linear system with continuous states $\mathcal{S} \in \mathbb{R}^{n_s}$ and actions $\mathcal{A} \in \mathbb{R}^{n_a}$ of form $T(s,a) = \mathbf{A}s + \mathbf{B}a + e$ and a reward function $r(s,a) = s^\top \mathbf{Q}s + a^\top \mathbf{R}a$ our task is to learn the optimal value function $v^*(s)$ and the optimal policy $\pi^*(s)$ by interacting with $T(s,a)$, where $\mathbf{A} \in \mathbb{R}^{n_s \times n_s}$ and $\mathbf{B} \in \mathbb{R}^{n_a \times n_s}$ are system matrices, $e \sim \mathcal{N}(0, \Sigma)$ is Gaussian noise, $\mathbf{Q} \in \mathbb{R}^{n_s \times n_s}$ is a negative semi-definite state reward matrix and $\mathbf{R} \in \mathbb{R}^{n_a \times n_a}$ is a negative definite action reward matrix.

It is well-known [76] that the optimal value and policy functions have the form:

$$v^*(s) = s^\top \mathbf{P}s + V_0, \qquad \pi^*(s) = \mathbf{K}s, \tag{C.1}$$

where $\mathbf{P} \in \mathbb{R}^{n_s \times n_s}$ is a negative semi-definite matrix, $\mathbf{K} \in \mathbb{R}^{n_a \times n_s}$ is a state feedback matrix, and $V_0 = \gamma(1-\gamma)^{-1}\text{Tr}\left(\mathbf{P}\Sigma\right)$.

Table C.5
Optimal hyper-parameters for supervised learning tasks.

Optimizer	Learning rate	Regularizer	Momentum	Line search	#CG iterations
California housing					
SGD	0.03	–	–	–	–
Adam	0.001	–	–	–	–
GAF	0.08	–	0.9	–	–
EGN	0.4	1.0	0.9	False	–
SGN	0.2	1.0	–	–	5
SQN	0.3	–	0.0	–	–
Superconduct					
SGD	0.0003	–	–	–	–
Adam	0.01	–	–	–	–
GAF	0.007	–	0.9	–	–
EGN	0.05	1.0	0.0	False	–
SGN	0.1	1.0	–	–	10
SQN	0.07	–	0.0	–	–
Diamonds					
SGD	2e-8	–	–	–	–
Adam	0.0005	–	–	–	–
GAF	0.001	–	0.0	–	–
EGN	0.0005	1.0	0.0	False	–
SGN	0.001	1.0	–	–	5
SQN	0.004	–	0.0	–	–
IMDB reviews					
SGD	0.005	–	–	–	–
Adam	0.01	–	–	–	–
GAF	0.003	–	0.9	–	–
EGN	0.01	1.0	0.0	False	–
SGN	0.05	1.0	–	–	5
SQN	0.02	–	0.0	–	–

Algorithm 6 Generalized policy iteration for LQR.

Input: initial stabilizing policy $\mathbf{K}_0$, initial weights $\mathbf{w}_0$, learning rate α, discount factor γ, tolerance $\eta = 10^{-8}$, LM regularizer ϵ.
Set policy iteration counter $p = 1$
repeat
  Given $\mathbf{K}_{p-1}$ estimate the corresponding weights $\mathbf{w}$ through a policy evaluation algorithm, e.g., Algorithm 7
  Convert weights $\mathbf{w}$ to a matrix $\mathbf{M}$
  if $\mathbf{M}_{aa}$ is not positive-definite **then**
    Return Error
  end if
  Improve policy $\mathbf{K}_p = -\mathbf{M}_{aa}^{-1}\mathbf{M}_{as}$
  Set $p = p + 1$
until $\left\|\mathbf{K}_p - \mathbf{K}_{p-1}\right\| < \eta$
Return $\mathbf{K}_p$

Algorithm 7 Policy evaluation for LQR.

Input: policy $\mathbf{K}$, initial weights $\mathbf{w}_0$, learning rate α, discount factor γ, tolerance $\eta = 10^{-8}$.
Set policy evaluation counter $i = 1$
Initialize S_1
repeat
  Choose action $A_i \sim \pi(S_i)$ by following an exploratory policy $\pi(s) = \mathbf{K}s + e$
  Execute action A_i and observe R_i, S_{i+1}
  Obtain A' by following a greedy policy $\pi(s) = \mathbf{K}s$
  Convert $[S_i, A_i]$ and $[S_{i+1}, A']$ to quadratic feature vectors $\mathbf{x}$ and $\mathbf{x}'$
  Calculate $\mathbf{d}_i = \left(R_i + \mathbf{w}_{i-1}^{\top}(\gamma\mathbf{x}' - \mathbf{x})\right)\mathbf{x}$
  Update weights $\mathbf{w}_i \leftarrow \mathbf{w}_{i-1} + \alpha_i\mathbf{d}_i$
  Set $i = i + 1$
until $\left\|\mathbf{w}_i - \mathbf{w}_{i-1}\right\|_{\infty} < \eta$
Return $\mathbf{w}_i$

To learn the optimal controller from data we can utilize Generalized Policy Iteration [75,89] (Algorithm 6).

BDT. System matrices for the model of a binary distillation tower (BDT) follow [78]. The matrices represent a continuous system. We discretize the system using a Zero-Order Hold (ZOH) method with a sampling rate of $\Delta T = 0.1s$.

UAV. System matrices for the linearized vertical plane dynamics of an aircraft (UAV) are taken from Ref. [79]. The matrices represent a continuous system. We discretize the system using a Zero-Order Hold (ZOH) method with a sampling rate of $\Delta T = 0.1s$.

Table C.6
Optimal hyper-parameters for BDT and UAV.

Optimizer	Learning rate	Regularizer	Momentum	Line search	#CG iterations
BDT					
SGD	0.0000005	–	–	–	–
Adam	0.1	–	–	–	–
EGN	1.0	1.0	0.0	False	–
SGN	1.0	1.0	–	–	10
UAV					
SGD	0.00008	–	–	–	–
Adam	0.02	–	–	–	–
EGN	0.2	1.0	0.0	False	–
SGN	1.0	1.0	–	–	10

Table C.7
Optimal hyper-parameters for Acrobot-v1 and Freeway-v1.

Optimizer	Learning rate	Regularizer	Momentum	Line search	#CG iterations
Acrobot-v1					
SGD	0.001	–	–	–	–
Adam	0.0003	–	–	–	–
EGN	0.1	1.0	0.0	False	–
SGN	0.005	1.0	–	–	3
Freeway-v1					
SGD	0.1	–	–	–	–
Adam	0.0003	–	–	–	–
EGN	0.4	1.0	0.0	False	–
SGN	0.5	1.0	–	–	5

The optimal sets of hyper-parameters for LQR are presented in Table C.6.

C.3. Reinforcement learning with DQN

Acrobot. `Acrobot-v1` is an OpenAI gym [81] environment where the goal is to swing the free end of the connected joints above a given height in as few steps as possible. Transitions to any non-terminal state yield reward $R_t = -1$.

Freeway. `Freeway-v1` is a part of MinAtar [82] package that emulates the original Atari Freeway game which plays out on a 10×10 grid. The goal is to reach the top of the screen starting at the bottom of the screen maneuvering the obstacles appearing on the screen. A reward of $+1$ is given upon reaching the top of the screen.

The optimal sets of hyper-parameters for reinforcement learning with DQN are presented in Table C.7.

C.4. Limitations

Table C.8

Table C.8
Wall-clock time (ms) per step for different batch sizes and MLP models, split into direction finding stage ("Solve") and remainder ("Other"). Means over 1000 runs (NVIDIA RTX A4000 GPU).

Batch size	MLP 1 K		MLP 10 K		MLP 100 K		MLP 1 M	
	Solve	Other	Solve	Other	Solve	Other	Solve	Other
8	0.135	0.120	0.177	0.118	0.176	0.120	0.492	0.274
16	0.156	0.114	0.169	0.111	0.219	0.144	0.629	0.429
32	0.165	0.112	0.196	0.113	0.304	0.183	0.943	0.725
64	0.223	0.099	0.265	0.111	0.398	0.235	1.958	1.351
128	0.326	0.106	0.381	0.149	0.734	0.376	3.489	2.576
256	0.555	0.098	0.716	0.197	1.397	0.623	10.035	5.459
512	1.381	0.140	1.781	0.258	4.220	0.861	38.511	6.239

Data availability

Experiments were conducted using publicly available datasets. Full details, including dataset sources, preprocessing steps, and experimental settings, are provided in Section 6 and Appendix C.

References

[1] H. Robbins, S. Monro, A stochastic approximation method, Ann. Math. Stat. (1951) 400–407.
[2] Y. Nesterov, A method of solving a convex programming problem with convergence rate $O(1/k^2)$, Dokl. Akad. Nauk SSSR 269 (3) (1983) 543.
[3] I. Sutskever, J. Martens, G. Dahl, G. Hinton, On the importance of initialization and momentum in deep learning, in: International Conference on Machine Learning, PMLR, 2013, pp. 1139–1147.
[4] J. Lucas, S. Sun, R. Zemel, R. Grosse, Aggregated momentum: stability through passive damping, arXiv preprint arXiv:1804.00325, 2018.
[5] J. Chen, C. Wolfe, Z. Li, A. Kyrillidis, Demon: improved neural network training with momentum decay, in: ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE, 2022, pp. 3958–3962.
[6] J. Duchi, E. Hazan, Y. Singer, Adaptive subgradient methods for online learning and stochastic optimization., J. Mach. Learn. Res. 12 (7) (2011).
[7] G. Hinton, N. Srivastava, K. Swersky, Neural networks for machine learning. Lecture 6a overview of mini-batch gradient descent, Cited On 14 (8) (2012) 2.
[8] M.D. Zeiler, Adadelta: an adaptive learning rate method, arXiv preprint arXiv:1212.5701, 2012.
[9] D.P. Kingma, J. Ba, Adam: a method for stochastic optimization, arXiv preprint arXiv:1412.6980, 2014.
[10] T. Dozat, Incorporating Nesterov momentum into Adam, (2016).
[11] M. Zaheer, S. Reddi, D. Sachan, S. Kale, S. Kumar, Adaptive methods for nonconvex optimization, Adv. Neural Inf. Process. Syst. 31 (2018).
[12] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 770–778.
[13] M. Naumov, D. Mudigere, H.-J.M. Shi, J. Huang, N. Sundaraman, J. Park, X. Wang, U. Gupta, C.-J. Wu, A.G. Azzolini, et al, Deep learning recommendation model for personalization and recommendation systems, arXiv preprint arXiv:1906.00091, 2019.
[14] T.B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D.M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, D. Amodei, Language models are few-shot learners, arXiv:2005.14165, 2020.
[15] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, et al, Llama: open and efficient foundation language models, arXiv preprint arXiv:2302.13971, 2023.
[16] I. Loshchilov, F. Hutter, Decoupled weight decay regularization, arXiv preprint arXiv:1711.05101, 2017.
[17] L. Sagun, U. Evci, V.U. Guney, Y. Dauphin, L. Bottou, Empirical analysis of the hessian of over-parametrized neural networks, arXiv preprint arXiv:1706.04454, 2017.
[18] P. Xu, F. Roosta, M.W. Mahoney, Second-order optimization for non-convex machine learning: an empirical study, in: Proceedings of the 2020 SIAM International Conference on Data Mining, SIAM, 2020, pp. 199–207.
[19] N. Agarwal, B. Bullins, E. Hazan, Second-order stochastic optimization for machine learning in linear time, J. Mach. Learn. Res. 18 (1) (2017) 4148–4187.
[20] R. Bollapragada, R.H. Byrd, J. Nocedal, Exact and inexact subsampled Newton methods for optimization, I.M.A. J. Numer. Anal. 39 (2) (2019) 545–578.
[21] Z. Yao, A. Gholami, S. Shen, M. Mustafa, K. Keutzer, M. Mahoney, Adahessian: an adaptive second order optimizer for machine learning, in: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 35, 2021, pp. 10665–10673.
[22] H. Liu, Z. Li, D. Hall, P. Liang, T. Ma, Sophia: a scalable stochastic second-order optimizer for language model pre-training, arXiv preprint arXiv:2305.14342, 2023.
[23] Y. Ren, D. Goldfarb, Efficient subsampled gauss-newton and natural gradient methods for training neural networks, arXiv preprint arXiv:1906.02353, 2019.
[24] N.N. Schraudolph, J. Yu, S. Günter, A stochastic quasi-Newton method for online convex optimization, in: Artificial Intelligence and Statistics, PMLR, 2007, pp. 436–443.
[25] R.H. Byrd, S.L. Hansen, J. Nocedal, Y. Singer, A stochastic quasi-Newton method for large-scale optimization, SIAM J. Optim. 26 (2) (2016) 1008–1031.
[26] A.S. Berahas, J. Nocedal, M. Takác, A multi-batch L-BFGS method for machine learning, Adv. Neural Inf. Process. Syst. 29 (2016).
[27] M. Gargiani, A. Zanelli, M. Diehl, F. Hutter, On the promise of the stochastic generalized gauss-newton method for training DNNs, arXiv preprint arXiv:2006.02409, 2020.
[28] Q. Tran-Dinh, N. Pham, L. Nguyen, Stochastic Gauss-Newton algorithms for nonconvex compositional optimization, in: International Conference on Machine Learning, PMLR, 2020, pp. 9572–9582.
[29] J.J. Brust, Nonlinear least squares for large-scale machine learning using stochastic Jacobian estimates, arXiv preprint arXiv:2107.05598, 2021.
[30] W.J. Duncan, Lxxviii. Some devices for the solution of large sets of simultaneous linear equations: with an appendix on the reciprocation of partitioned matrices, Lond., Edinb., Dubl. Philos. Mag. J. Sci. 35 (249) (1944) 660–670.
[31] L. Guttman, Enlargement methods for computing the inverse matrix, Ann. Math. Stat. (1946) 336–343.
[32] J. Martens, et al, Deep learning via Hessian-free optimization., in: ICML, vol. 27, 2010, pp. 735–742.
[33] J. Martens, I. Sutskever, Learning recurrent neural networks with Hessian-free optimization, in: Proceedings of the 28th International Conference on Machine Learning (ICML-11), 2011, pp. 1033–1040.
[34] R. Kiros, Training neural networks with stochastic Hessian-free optimization, arXiv preprint arXiv:1301.3641, 2013.
[35] O. Vinyals, D. Povey, Krylov subspace descent for deep learning, in: Artificial Intelligence and Statistics, PMLR, 2012, pp. 1261–1268.
[36] A.G. Wills, T.B. Schön, Stochastic quasi-Newton with line-search regularisation, Automatica 127 (2021) 109503.
[37] C. Liu, L. Luo, Quasi-Newton methods for saddle point problems, Adv. Neural Inf. Process. Syst. 35 (2022) 3975–3987.
[38] A. Botev, H. Ritter, D. Barber, Practical Gauss-Newton optimisation for deep learning, in: International Conference on Machine Learning, PMLR, 2017, pp. 557–565.
[39] S.-I. Amari, Natural gradient works efficiently in learning, Neural Comput. 10 (2) (1998) 251–276.
[40] F. Kunstner, P. Hennig, L. Balles, Limitations of the empirical Fisher approximation for natural gradient descent, Adv. Neural Inf. Process. Syst. 32 (2019).
[41] L. Bottou, F.E. Curtis, J. Nocedal, Optimization methods for large-scale machine learning, SIAM Rev. 60 (2) (2018) 223–311.
[42] S. Sun, Z. Cao, H. Zhu, J. Zhao, A survey of optimization methods from a machine learning perspective, IEEE Trans. Cybern. 50 (8) (2019) 3668–3681.
[43] O. Pooladzandi, Y. Zhou, Improving levenberg-Marquardt algorithm for neural networks, arXiv preprint arXiv:2212.08769, 2022.
[44] A.D. Adeoye, A. Bemporad, SC-Reg: Training Overparameterized Neural Networks under Self-Concordant Regularization, IMT School for Advanced Studies Lucca (2021).
[45] A.D. Adeoye, A. Bemporad, SCORE: Approximating curvature information under self-concordant regularization, Comput. Optim. Appl. 86 (2) (2023) 599–626.
[46] N. Doikov, M. Jaggi, et al, Second-order optimization with lazy hessians, in: International Conference on Machine Learning, PMLR, 2023, pp. 8138–8161.
[47] F.E. Curtis, K. Scheinberg, Adaptive stochastic optimization: a framework for analyzing stochastic optimization algorithms, IEEE Signal Process. Mag. 37 (5) (2020) 32–42.
[48] Y. Hong, H. Bergou, N. Doucet, H. Zhang, J. Cranney, H. Ltaief, D. Gratadour, F. Rigaut, D.E. Keyes, Stochastic Levenberg-Marquardt for Solving Optimization Problems on Hardware Accelerators, Submitted to IEEE, 2020.
[49] K.P. Murphy, Probabilistic Machine Learning: an Introduction, MIT Press, 2022, probml.ai
[50] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, Ł. Kaiser, I. Polosukhin, Attention is all you need, Adv. Neural Inf. Process. Syst. 30 (2017).
[51] xai-org, Grok-1, gitHub repository (2024) https://github.com/xai-org/grok-1
[52] A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Yang, A. Fan, et al, The llama 3 herd of models, arXiv preprint arXiv:2407.21783, 2024.
[53] V. Papyan, The full spectrum of deep net hessians at scale: dynamics with sample size, arXiv preprint arXiv:1811.07062, 2018.
[54] N.N. Schraudolph, Fast curvature matrix-vector products for second-order gradient descent, Neural Comput. 14 (7) (2002) 1723–1738.
[55] A.R. Sankar, Y. Khasbage, R. Vigneswaran, V.N. Balasubramanian, A deeper look at the Hessian eigenspectrum of deep neural networks and its applications to regularization, in: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 35, 2021, pp. 9481–9488.
[56] J. Nocedal, S. Wright, Numerical Optimization, Springer Series in Operations Research and Financial Engineering, Springer New York, 2006.
[57] M. Arbel, R. Menegaux, P. Wolinski, Rethinking Gauss-Newton for learning over-parameterized models, Adv. Neural Inf. Process. Syst. 36 (2024).
[58] N.J. Higham, Accuracy and Stability of Numerical Algorithms, SIAM, 2002.
[59] F. Zhang, M. Pilanci, Optimal shrinkage for distributed second-order optimization, in: International Conference on Machine Learning, PMLR, 2023, pp. 41523–41549.
[60] O. Chapelle, D. Erhan, et al, Improved preconditioner for Hessian free optimization, in: NIPS Workshop on Deep Learning and Unsupervised Feature Learning, vol. 201, Citeseer, 2011.
[61] S. Vaswani, A. Mishkin, I. Laradji, M. Schmidt, G. Gidel, S. Lacoste-Julien, Painless stochastic gradient: interpolation, line-search, and convergence rates, Adv. Neural Inf. Process. Syst. 32 (2019).
[62] R. Johnson, T. Zhang, Accelerating stochastic gradient descent using predictive variance reduction, Adv. Neural Inf. Process. Syst. 26 (2013).
[63] A. Defazio, F. Bach, S. Lacoste-Julien, Saga: a fast incremental gradient method with support for non-strongly convex composite objectives, Adv. Neural Inf. Process. Syst. 27 (2014).
[64] L.M. Nguyen, J. Liu, K. Scheinberg, M. Takáč, Sarah: a novel method for machine learning problems using stochastic recursive gradient, in: International Conference on Machine Learning, PMLR, 2017, pp. 2613–2621.
[65] C. Paquette, K. Scheinberg, A stochastic line search method with expected complexity analysis, SIAM J. Optim. 30 (1) (2020) 349–376.
[66] S. Ghadimi, G. Lan, Stochastic first-and zeroth-order methods for nonconvex stochastic programming, SIAM J. Optim. 23 (4) (2013) 2341–2368.
[67] R.M. Gower, M. Schmidt, F. Bach, P. Richtárik, Variance-reduced methods for machine learning, Proc. IEEE 108 (11) (2020) 1968–1983.
[68] A. Defazio, L. Bottou, On the ineffectiveness of variance reduced optimization for deep learning, Adv. Neural Inf. Process. Syst. 32 (2019).
[69] M. Liu, L. Chen, X. Du, L. Jin, M. Shang, Activated gradients for deep neural networks, IEEE Trans. Neural Netw. Learn. Syst. 34 (4) (2021) 2156–2168.
[70] R.K. Pace, R. Barry, Sparse spatial autoregressions, Stat. Probab. Lett. 33 (3) (1997) 291–297.

[71] K. Hamidieh, Superconductivty data, UCI Machine Learning Repository, DOI: 2018 https://doi.org/10.24432/C53P47
[72] H. Wickham, Ggplot2: Elegant Graphics for Data Analysis, Springer-Verlag New York, 2016, https://ggplot2.tidyverse.org
[73] A.L. Maas, R.E. Daly, P.T. Pham, D. Huang, A.Y. Ng, C. Potts, Learning word vectors for sentiment analysis, in: Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, Association for Computational Linguistics, Portland, Oregon, USA, 2011, pp. 142–150, http://www.aclweb.org/anthology/P11-1015
[74] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, M. Riedmiller, Playing atari with deep reinforcement learning, arXiv preprint arXiv:1312.5602, 2013.
[75] S.J. Bradtke, B.E. Ydstie, A.G. Barto, Adaptive linear quadratic control using policy iteration, in: Proceedings of 1994 American Control Conference-ACC'94, vol. 3, IEEE, 1994, pp. 3475–3479.
[76] E. Hazan, K. Singh, Introduction to online nonstochastic control, arXiv preprint arXiv:2211.09619, 2022.
[77] F. Leibfritz, Compleib: Constrained matrix optimization problem library, (2006).
[78] E.J. Davison, Benchmark problems for control system design, Rep. I.F.A.C. Theory Comm. (1990).
[79] Y.S. Hung, A.G.J. MacFarlane, Multivariable control: a quasiclassical approach, (1982).
[80] S. Huang, R.F.J. Dossa, C. Ye, J. Braga, Cleanrl: high-quality single-file implementations of deep reinforcement learning algorithms, arXiv preprint arXiv:2111.08819, 2021.
[81] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, W. Zaremba, OpenAI Gym, (2016). arXiv:arXiv:1606.01540.
[82] K. Young, T. Tian, Minatar: an atari-inspired testbed for thorough and reproducible reinforcement learning experiments, arXiv preprint arXiv:1903.03176, 2019.
[83] R. Grosse, Taylor approximations, Neural Netw. Train. Dyn. Lect. Notes Univ. Toronto (2021).
[84] B.M. Ozyildirim, M. Kiran, Levenberg–Marquardt multi-classification using hinge loss function, Neural Netw. 143 (2021) 564–571.
[85] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: machine learning in Python, J. Mach. Learn. Res. 12 (2011) 2825–2830.
[86] Q. Lhoest, A. Villanova Del Moral, Y. Jernite, A. Thakur, P. von Platen, S. Patil, J. Chaumond, M. Drame, J. Plu, L. Tunstall, J. Davison, M. Šaško, G. Chhablani, B. Malik, S. Brandeis, T. Le Scao, V. Sanh, C. Xu, N. Patry, A. McMillan-Major, P. Schmid, S. Gugger, C. Delangue, T. Matussière, L. Debut, S. Bekman, P. Cistac, T. Goehringer, V. Mustar, F. Lagunas, A. Rush, T. Wolf, Datasets: a community library for natural language processing, in: Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, Association for Computational Linguistics, Online and Punta Cana, Dominican Republic, 2021, pp. 175–184, https://aclanthology.org/2021.emnlp-demo.21
[87] TensorFlow, TensorFlow datasets, a collection of ready-to-use datasets, https://www.tensorflow.org/datasets
[88] M. Honnibal, I. Montani, S. Van Landeghem, A. Boyd, et al, Spacy: industrial-strength natural language processing in Python, (2020).
[89] S.J. Bradtke, A.G. Barto, Linear least-squares algorithms for temporal difference learning, Machine Learn. 22 (1) (1996) 33–57.

Author biography

Mikalai Korbit received his Master's degree in Computer Science from the Georgia Institute of Technology, USA. He is currently pursuing a Ph.D. in the Dynamical Systems, Control, and Optimization (DYSCO) group at the IMT School for Advanced Studies Lucca, Italy. Previously, he was a visiting researcher in the Intelligent Robotics group at Aalto University, Finland. His research is focused on developing scalable second-order optimization methods for large-scale deep neural networks.

Adeyemi D. Adeoye received the bachelor's degree (Hons.) in mathematics from the University of Ilorin, Ilorin, Nigeria, in 2016, the master's degree in mathematical sciences from the African Institute for Mathematical Sciences, Limbe, Cameroon, in 2018, and the master's degree in machine intelligence from the African Institute for Mathematical Sciences, Kigali, Rwanda, in 2021. He obtained his Ph.D. degree with the Dynamical Systems, Control, and Optimization Research Unit, IMT School for Advanced Studies Lucca, Lucca, Italy. His research interests include mathematical optimization, data-driven control, and neural networks.

Alberto Bemporad received his Master's degree cum laude in Electrical Engineering in 1993 and his Ph.D. in Control Engineering in 1997 from the University of Florence, Italy. In 1996/97 he was with the Center for Robotics and Automation, Department of Systems Science & Mathematics, Washington University, St. Louis. In 1997-1999 he held a postdoctoral position at the Automatic Control Laboratory, ETH Zurich, Switzerland, where he collaborated as a Senior Researcher until 2002. In 1999-2009 he was with the Department of Information Engineering of the University of Siena, Italy, becoming an Associate Professor in 2005. In 2010-2011 he was with the Department of Mechanical and Structural Engineering of the University of Trento, Italy. Since 2011 he has been a Full Professor at the IMT School for Advanced Studies Lucca, Italy, where he served as the Director of the institute from 2012 to 2015. He spent visiting periods at Stanford University, the University of Michigan, and Zhejiang University. In 2011 he co-founded ODYS S.r.l., a company specialized in developing model predictive control systems for industrial production. He has published more than 400 papers in the areas of model predictive control, hybrid systems, optimization, and automotive control, and is the co-inventor of 21 patents. He is the author or coauthor of various software packages for model predictive control design and implementation, including the Model Predictive Control Toolbox (The Mathworks, Inc.) and the Hybrid Toolbox for MATLAB. He was an Associate Editor of the IEEE Transactions on Automatic Control during 2001-2004 and Chair of the Technical Committee on Hybrid Systems of the IEEE Control Systems Society from 2002 to 2010. He received the IFAC High-Impact Paper Award for the 2011-14 triennial, the IEEE CSS Transition to Practice Award in 2019, the 2021 SAE Environmental Excellence in Transportation Award, the 2024 Beale–Orchard-Hays Prize for Excellence in Computational Mathematical Programming, and an ERC Advanced Research Grant in 2024. He has been an IEEE Fellow since 2010.

Mario Zanon received the Master's degree in Mechatronics from the University of Trento and the Diplôme d'Ingénieur from the École Centrale Paris, in 2010. After research stays at the KU Leuven, University of Bayreuth, Chalmers University, and the University of Freiburg he received the Ph.D. degree in Electrical Engineering from the KU Leuven in November 2015. He held a Post-Doc researcher position at Chalmers University until the end of 2017. In 2018 he became Assistant Professor at the IMT School for Advanced Studies Lucca where he became Associate Professor in 2021. His research interests include numerical methods for optimization, economic MPC, reinforcement learning, and the optimal control and estimation of nonlinear dynamic systems, in particular for aerospace and automotive applications.