EXACT STOCHASTIC NEWTON METHOD FOR DEEP LEARNING: THE FEEDFORWARD NETWORKS CASE.

Anonymous authors
Paper under double-blind review

ABSTRACT

The inclusion of second-order information into Deep Learning optimization has drawn consistent interest as a way forward to improve upon gradient descent methods. Estimating the second-order update is often convoluted and computationally expensive, which drastically limits its usage scope and forces the use of various truncations and approximations. This work demonstrates that it is possible to solve the Newton direction in the stochastic case exactly. We consider feedforward networks as a base model, build a second-order Lagrangian which we call Sifrian, and provide a closed-form formula for the exact stochastic Newton direction under some monotonicity and regularization conditions. We propose a convexity correction to escape saddle points, and we reconsider the intrinsic stochasticity of the online learning process to improve upon the formulas. We finally compare the performance of the developed solution with well-established training methods and show its viability as a training method for Deep Learning.

Optimization in Deep Learning is mainly dominated by first-order methods built around the central concept of backpropagation (LeCun et al., 1988). Second-order methods have exceptional theoretical properties in deterministic optimization (Boyd et al., 2004; Nocedal & Wright, 2006), but such properties do not translate well into the stochastic case (LeCun et al., 2012; Bottou et al., 2018). First-order methods such as the Stochastic Gradient Descent (SGD) (Robbins & Monro, 1951) are relatively simple and have been adaptively improved for Deep Learning (Duchi et al., 2011; Tieleman & Hinton, 2012; Kingma & Ba, 2014; Reddi et al., 2019; Yao et al., 2020). Inherently, second-order methods might seems inadequate for Deep learning due to increased computational cost, poor clock-wall performance, and the non-convex nature of Deep Learning (LeCun et al., 2012). Furthermore, Newton method might even reduce the generalization capabilities of training (Wadia et al., 2021; Amari et al., 2020). Despite these various limitations, substantial effort was deployed to include Hessian information into the optimization process (e.g. Byrd et al., 2011; Sohl-Dickstein et al., 2014; Byrd et al., 2016; Agarwal et al., 2017; Berahas et al., 2019; Anil et al., 2020; Goldfarb et al., 2020; Castera et al., 2021). Several approaches exist such as the Gauss-Newton method (e.g Schraudolph, 2002; Botev et al., 2017), diagonal approximation of the Hessian (e.g. Bordes et al., 2009; Schaul et al., 2013), iterative low-rank updates such as BFGS (Broyden, 1970; Fletcher, 1970; Goldfarb, 1970; Shanno, 1970) (see also, Liu & Nocedal, 1989; Schraudolph et al., 2007; Bollapragada et al., 2018), or Hessian-Free methods which combine the fast Hessian matrix multiplication (Pearlmutter, 1994) and the conjugate gradient algorithm (e.g. Martens, 2010; Martens & Sutskever, 2012; Dauphin et al., 2014). The use of the Fisher matrix to capture curvature information in the space of distributions, instead of the Hessian, is another approach which yields the natural gradient (Amari, 1998), but suffers from the same computational issues as the Newton method. Within this context K-FAC method (Martens & Grosse, 2015; Ba et al., 2016a; George et al., 2018) mitigates some of the computational issues of the natural gradient method. In essence, several drawbacks and flaws limits the adoption of second-order methods as a standard for neural networks training .

In this paper, we delve into the basics of neural networks training and propose a novel approach to characterize the Newton update which help us derive an exact closed-form solution for the stochastic Newton method. Our method requires a suitable regularization of the neural network and strict monotonicity of the activation functions. We start this paper by introducing useful notations while deriving the well-known backpropagation algorithm for a feedforward network. Then we introduce a second-order Lagrangian which we call Sifrian, that will serve to characterize the Newton direction. We derive four types of equations from the Sifrian, and we provide an exact closed-form solution

for the Newton direction in the stochastic case. We further propose a saddle-free version of our method, and add a randomisation process to enhance our solution. In the last part of this paper, we show the applicability of our method for Deep Learning through diverse classification tasks using feedforward architectures.

1 PRELIMINARIES

Deep Learning and neural networks training could be seen as an optimization problem of an expected loss function ℓ over a distribution $\mathcal{D}$ of labeled patterns (x, d). In general, weights and biases (W, β) are the sought-after parameters of the network. The labeled patterns distribution $\mathcal{D}$ is often unknown, but a large number of samples (database D) allows the approximation of the expected loss with an empirical risk:

$$\min_{W,\beta} \left(\mathbb{E}_{(x,d)\sim\mathcal{D}}\left[\ell\left(W,\beta,(x,d)\right)\right]\right) \rightarrow \frac{1}{|D|}\sum_{p\in D}\ell\left(W,\beta,(x_p,d_p)\right), \quad (x_p,d_p)_{p\in D} \overset{i.i.d}{\sim} \mathcal{D}. \tag{1}$$

The loss or cost function ℓ is typically a cross entropy function or an l^2 norm of the mismatches between the network outputs and the labels: $\ell\left(W,\beta,(x_p,d_p)\right) = \frac{1}{2}\left\|x_p(n) - d_p\right\|_2^2$. Where n indicates the output layer number of the network. We present more in detail the architecture of a Feedforward Neural Network (FNN) hereafter.

1.1 NOTATIONS AND FEEDFORWARD NEURAL NETWORKS (FNN)

In this section, we recall the main details of FNNs, which will be used as a standard model for Deep Learning[1]. The notation used throughout this paper is similar to notations presented in LeCun et al. (1988). The main equation governing the FNN, a.k.a. the forward model is the following:

$$x_p(k) = F\left(W(k)\,x_p(k-1) + \beta(k)\right), \quad k = 1..n, \quad p \in D. \tag{2}$$

where D is the database, p designs one single pattern from the database (e.g. one single image or audio recording), k is the layer number and n is the total number of layers in the network. The initial input for pattern p is $x_p(0)$ (e.g. the vectorized input image data). The state variable x_p is transformed at each layer k through a multiplication by a weight matrix $W(k)$ and an addition of a bias vector $\beta(k)$. The activation function F, which is typically a sigmoid or a Rectified Linear Unit (ReLU), is applied element wise on the resulting vector $a_p(k) = W(k)\,x_p(k-1) + \beta(k)$ and serves to introduce non-linearity in the neural network.

Compact notation: Some of the equations in this paper are long. For the sake of clarity we will use a compact notation when needed e.g. $W(k) \rightarrow W_k$, $x_p\left(k\right) \rightarrow x_{p,k}$ or $\nabla F\left(a_p(k)\right) \rightarrow \nabla F_{p,k}$.

1.2 THE LAGRANGIAN AND BACKPROPAGATION

The origin of backpropagation could be traced back to the early 1970s and could be derived by casting Deep Learning as a constrained optimization problem. This section is similar to LeCun et al. (1988). The main novelty in this section is the addition of regularization term dependent on the state variable $R(x)$ to the cost function ℓ: $\ell_R = \ell + R\left(x\right)$. The regularization has to verify the admissibility criteria which we define hereafter:

Definition 1: A regularization term is admissible if the second derivative of the augmented cost function w.r.t. to the state variable is cross-talk free and invertible, i.e.:

$$\forall\left(p,q\right)\in D,\, \forall\left(k,m\right)\in\left[1..n\right] \quad \Lambda = \left(\frac{\partial \ell_R}{\partial x_p(k)\partial x_q(m)}\right)_{p,q,k,m} \propto \delta_{p=q}. \tag{3}$$

Where δ is the Kronecker delta. An example of such regularization is the following function:

$$R(x) = \frac{1}{2}\sum_{p\in D}\sum_{k=1..n-1}\left\langle x_p(k), \Lambda_p\left(k\right)x_p(k)\right\rangle \tag{4}$$

[1]Although we only train FNN, the same concepts still apply to convolutional networks.

$\langle .,. \rangle$ is the canonical inner product and $(\Lambda_p)_{p,k}$ are positive symmetric matrices. Such a choice might seem atypical since the regularization often concerns the network parameters, mainly the weights or the biases. This "admissible" regularization was introduced for the sole purpose of guaranteeing an invertible curvature matrix Λ which could be further defined separately for each pattern i.e. Λ_p. We will show later that such a regularization minimizes the norm of the gradient (Barrett & Dherin, 2020; Smith et al., 2021). In order to derive the backpropagation algorithm we introduce the following Lagrangian as per the notation of LeCun et al. (1988):

$$\mathcal{L}(x, W, \beta, b) = \ell_R + \sum_{p \in D} \sum_{k=1..n} \langle x_p(k) - F\left(W(k)\, x_p(k-1) + \beta(k)\right), b_p(k) \rangle \,. \tag{5}$$

The Lagrangian contains the original cost function, the admissible regularization term, and the product of the forward equation with the adjoint state vectors $(b_p(k))_{p,k}$. These adjoint vectors are defined for each layer of the network and each pattern of the database; however, their values are not fixed yet and will be chosen in a way that simplifies the gradient computation. If the state variable x verifies the forward equations, then the Lagrangian simplifies to the cost function:

$$\mathcal{L}(x(W, \beta), W, \beta, b) = \ell_R(x(W, \beta)). \tag{6}$$

The total derivative of the previous Lagrangian w.r.t. the weights or biases is the gradient and could be expressed in terms of partial derivatives as follows[2]:

$$\begin{cases} \frac{d\mathcal{L}(x(W,\beta),W,\beta,b)}{dW(k)} = \frac{\partial \mathcal{L}}{\partial W(k)} + \sum_{p \in D} \sum_{m=1..n} \left(\frac{dx_p(m)}{dW(k)}\right) \left(\frac{\partial \mathcal{L}}{\partial x_p(m)}\right) = \frac{d\ell_R}{dW(k)}, \\ \\ \frac{d\mathcal{L}(x(W,\beta),W,\beta,b)}{d\beta(k)} = \frac{\partial \mathcal{L}}{\partial \beta(k)} + \sum_{p \in D} \sum_{m=1..n} \left(\frac{dx_p(m)}{d\beta(k)}\right) \left(\frac{\partial \mathcal{L}}{\partial x_p(m)}\right) = \frac{d\ell_R}{d\beta(k)}. \end{cases} \tag{7}$$

The partial derivatives of the Lagrangian w.r.t to $\{W, \beta\}_{k=1..n}$ are straightforward to compute. The Fréchet derivatives $\left(\frac{dx_p(m)}{dW(k)}, \frac{dx_p(m)}{d\beta(k)}\right)_{p,k,m}$ are non-trivial to evaluate and the core idea of backpropagation is the selection of the adjoint state variables $(b_p(k))_{p,k}$ which cancels the superfluous terms. Such a simplification is achievable if:

$$\left(\frac{\partial \mathcal{L}}{\partial x_p(k)}\right)_{p \in D,\, k=1..n} = 0. \tag{8}$$

Computing the previous partial derivative yields the following backpropagation equation:

$$\frac{\partial \mathcal{L}}{\partial x_p(k)} = b_p(k) - \mathbf{1}_{k=1..n-1} W(k+1)^T \nabla F\left(a_p(k+1)\right) b_p(k+1) + \frac{\partial \ell_R}{\partial x_p(k)} = 0. \tag{9}$$

$\mathbf{1}_E$ is the indicator function, and it is equal to one if the underlying condition E is true and null otherwise. The nabla operator ∇F is a diagonal square matrix with an element-wise derivative of its argument (along the diagonal). We added the activation variable $a_p(k) = W(k)x_p(k-1) + \beta(k)$ to further simplify the notations. The resolution of the backpropagation system could be split into a boundary condition and a backward propagation system. Further details could be found in LeCun et al. (1988). The particular choice of the adjoint state vectors $(b_p(k))_{p \in D,\, k=1..n}$ yields a simple formula for the gradient of the cost function:

$$\begin{cases} G(k) = \frac{\partial \mathcal{L}}{\partial W(k)} = -\sum_p \nabla \left[F\left(a_p(k)\right)\right] b_p(k) x_p(k-1)^T, \\ \\ g(k) = \frac{\partial \mathcal{L}}{\partial \beta(k)} = -\sum_p \nabla F\left(a_p(k)\right) b_p(k). \end{cases} \tag{10}$$

The backpropagation algorithm is fundamental for Deep Learning. It is simple and provides a straightforward solution to an otherwise tedious problem to solve. Unfortunately, developing an efficient second-order backpropagation method remains elusive. The current state-of-the-art is based on the use of the $\mathcal{R}-$operator (Pearlmutter, 1994) to compute the product of the Hessian with a given vector without explicitly calculating or storing the Hessian (Martens, 2010; Dauphin et al., 2014; Agarwal et al., 2017). In the following section, we provide a different framework for backpropagation which allows the characterization of the second-order Newton direction.

[2]A denominator layout notation for derivation is used throughout this paper.

2 THE SIFRIAN AND THE EXACT NEWTON DIRECTION

It is legitimate to wonder if the Lagrangian multipliers method could be replicated for higher-order optimization. We hereafter provide a solution for the second-order case and build an equivalent Lagrangian which we call the Sifrian.

2.1 THE SIFRIAN

The Lagrangian is useful to derive the first-order gradient, and it combines the cost function and the forward model multiplied by adjoint variables. Our approach to extend the Lagrangian consists of creating a function with the forward model, the backward model, and the definition of the gradient. We choose to omit the loss function, as its derivation will generate the gradient again. Starting from this paradigm, we introduce the Sifrian[3].

Definition 2: The Sifrian $\mathcal{S}$ of the feed-forward network is defined as follows:

$$\begin{aligned}
&\mathcal{S}(x, b, W, \beta, G, g, \gamma, \zeta, N, \eta) = \textstyle\sum_{k,p} \langle x_p(k) - F\left(a_p(k)\right), \gamma_p(k)\rangle \\
&+ \left\langle b_p(k) - \mathbf{1}_{k=1..n-1} W(k+1)^T \nabla F\left(a_p(k+1)\right) b_p(k+1) + \tfrac{\partial \ell_R}{\partial x_p(k)}, \zeta_p(k) \right\rangle \\
&+ \textstyle\sum_k \left\langle G(k) + \sum_p \nabla\left[F\left(a_p(k)\right)\right] b_p(k) x_p(k-1)^T, N(k) \right\rangle \\
&+ \left\langle g(k) + \textstyle\sum_p \nabla F\left(a_p(k)\right) b_p(k), \eta(k) \right\rangle.
\end{aligned} \tag{11}$$

The Sifrian includes all the equations generated by the first order backpropagation, multiplied by four new adjoint parameters $\{\gamma_{p,k}, \zeta_{p,k}, N_k, \eta_k\}$. While the Lagrangian describes a saddle point (Nocedal & Wright, 2006), the Sifrian, describes an equilibrium and it has a unique null value when all the forward, backward and gradients equations are verified i.e.:

$$\mathcal{S}(x(W,\beta), W, \beta, b(W,\beta,x), G(W,\beta,x,b), \gamma, \zeta, N, \eta) = 0. \tag{12}$$

We call the previous quantity the equilibrated Sifrian, and it is critical to notice its invariance to total derivation w.r.t. either the weights or biases (i.e., remains null). Expressing the total derivative of the equilibrated Sifrian using partial derivatives yields the following equations:

$$\begin{cases}
\frac{d\mathcal{S}}{dW_k} = \frac{\partial \mathcal{S}}{\partial W_k} + \sum_{m,p} \frac{dx_{p,m}}{dW_k} \frac{\partial \mathcal{S}}{\partial x_{p,m}} + \frac{db_{p,m}}{dW_k} \frac{\partial \mathcal{S}}{\partial b_{p,m}} + \sum_m \frac{dG_m}{dW_k} \frac{\partial \mathcal{S}}{\partial G_m} + \frac{dg_m}{dW_k} \frac{\partial \mathcal{S}}{\partial g_m} = 0, \\
\frac{d\mathcal{S}}{d\beta_k} = \frac{\partial \mathcal{S}}{\partial \beta_k} + \sum_{m,p} \frac{dx_{p,m}}{d\beta_k} \frac{\partial \mathcal{S}}{\partial x_{p,m}} + \frac{db_{p,m}}{d\beta_k} \frac{\partial \mathcal{S}}{\partial b_{p,m}} + \sum_m \frac{dG_m}{d\beta_k} \frac{\partial \mathcal{S}}{\partial G_m} + \frac{dg_m}{d\beta_k} \frac{\partial \mathcal{S}}{\partial g_m} = 0.
\end{cases} \tag{13}$$

Similar to first order backpropagation, we select the new adjoints variables $\{\gamma_{p,k}, \zeta_{p,k}, N_k, \eta_k\}_{p \in D,\, k=1..n}$ to circumvent the calculation of any superfluous terms. We make the following choice:

$$\forall\, m, \forall\, q: \quad \frac{\partial \mathcal{S}}{\partial W(m)} = -G(m), \quad \frac{\partial \mathcal{S}}{\partial \beta(m)} = -g(m), \quad \frac{\partial \mathcal{S}}{\partial x_q(m)} = 0, \quad \frac{\partial \mathcal{S}}{\partial b_q(m)} = 0. \tag{14}$$

In this case, after vectorizing the weight gradient $G(k)$ (and also $N(k)$), and concatenating all their values across all layers, the Equ. 13 could be cast as the following block matrix system:

$$\begin{bmatrix} \frac{dG}{dW} & \frac{dg}{dW} \\ \frac{dG}{d\beta} & \frac{dg}{d\beta} \end{bmatrix} \begin{bmatrix} N \\ \eta \end{bmatrix} = \begin{bmatrix} G \\ g \end{bmatrix} \rightarrow H \begin{bmatrix} N \\ \eta \end{bmatrix} = \begin{bmatrix} G \\ g \end{bmatrix}. \tag{15}$$

The matrix that appears in the previous system is nothing else but the Hessian. Hence the computation of the Newton direction could be achieved by solving the associated system of four equations in Equ. 14. The derivation of the four types of Sifrian equations[4] is provided with further details in

[3]The choice of the Sifrian appellation stems from the Arabic word "Sifr", which means zero. It is also the characteristic value of the Sifrian when all the equations of Deep Learning are satisfied.

[4]The Sifrian yields a total of $2\,|D| + 2$ equations with an equal number of unknowns.

the Appendix (A.1). Using the compact notation, we report hereafter the equations for piece-wise affine activation functions (e.g., ReLU or Leaky ReLU). The general case of $\mathcal{C}^2$ activation function is included in the Appendix (A.1).

$$
\begin{aligned}
&(\mathbf{1})\ \textstyle\sum_p \nabla F_{p,k}\left(\gamma_{p,k} x_{p,k-1}^T + b_{p,k} x_{p,k-1}^T + \mathbf{1}_{k=2..n} b_{p,k} \zeta_{p,k-1}^T\right) = 0,\\
&(\mathbf{2})\ \textstyle\sum_p \nabla F_{p,k}\left(\gamma_{p,k} + b_{p,k}\right) = 0,\\
&(\mathbf{3})\ \gamma_{p,k} - \mathbf{1}_{k<n} W_{k+1}^T \nabla F_{p,k+1} \gamma_{p,k+1} + \textstyle\sum_m \frac{\partial^2 \ell_R}{\partial x_{p,k} \partial x_{p,m}} \zeta_{p,m} + \mathbf{1}_{k<n} N_{k+1}^T \nabla F_{p,k+1} b_{p,k+1} = 0,\\
&(\mathbf{4})\ \zeta_{p,k} - \mathbf{1}_{k=2..n} \nabla F_{p,k} W_k \zeta_{p,k-1} + \nabla F_{p,k}\left(N_k x_{p,k-1} + \eta_k\right) = 0.
\end{aligned}
\tag{16}
$$

2.2 THE EXACT STOCHASTIC NEWTON (ESN)

The resolution of the previous system is intricate in the general case. However, if we consider a single pattern, i.e., the stochastic case and a strictly monotonous activation function, then a closed-form solution could be derived.

Main theorem: If the cost function is admissibly regularized, the activation function F is strictly-monotonous piece-wise affine, and the first order adjoints $(b_p(k))_{p,k}$ are non-null, then the minimal-rank stochastic Newton direction has a closed-form solution:

$$
\begin{cases}
N(k) = -\frac{\nabla F(a_p(k)) b_p(k)}{\|\nabla F(a_p(k)) b_p(k)\|_2^2} \left(\frac{\partial \ell_R}{\partial x_p(k-1)}\right)^T, \\
\eta(k) = \mathbf{1}_{k=n} \left[\nabla F\left(a_p(n)\right)\right]^{-1} \left\{\frac{\partial^2 \ell_R}{\partial x_p(n)^2}\right\}^{-1} \frac{\partial \ell_R}{x_p(n)} - N(k)\, x_p(k-1).
\end{cases}
\tag{17}
$$

Proof: Under the proposition assumptions, the Sifrian second equation simplifies to $\gamma_p(k) = -b_p(k)$, which starts a cascade of simplifications. The first equations becomes $\mathbf{1}_{k=1..n-1}\zeta_p(k-1) = 0$. The last layer of the third equation yields: $\gamma_p(n) + \frac{\partial^2 \ell_R}{\partial x_p(n)^2}\zeta_p(n) = 0$. The remaining two equations become:

$$
\begin{aligned}
&\mathbf{1}_{k=1..n-1} \tfrac{\partial \ell_R}{x_p(k)} + \mathbf{1}_{k=1..n-1} N(k+1)^T \nabla\left[F\left(a_p(k+1)\right)\right] b_p(k+1) = 0,\\
&-\mathbf{1}_{k=n} \left\{\tfrac{\partial^2 \ell_R}{\partial x_p(n)^2}\right\}^{-1} \tfrac{\partial \ell_R}{x_p(n)} + \nabla\left[F\left(a_p(k)\right)\right]\left[N(k)\, x_p(k-1) + \eta(k)\right] = 0.
\end{aligned}
\tag{18}
$$

The Newton weight is a matrix, and it is determined through its product with the adjoint vector: so it is only characterized through one direction, hence determining the Newton direction is a degenerate problem, but the minimal rank solution is still unique, and it is given in Equ. 17. One issue still remains: the results of the Newton weight do not extend to the first layer (Equ. 18). To circumvent this shortcoming, we introduce an additional layer before the input, which corresponds to the identity multiplication and a null bias. This layer remains unchanged throughout the optimization process, and we call it the "white layer".This pre-input manipulation allows the extension of the solution to the first layer. □

White layer concept The third equation of the Sifrian allows the characterization of the Newton weight $N(k)$ only between the 2nd and the output (nth) layers. Such a problem is **fundamental** and it is legitimate to wonder how is it possible for Hessian-Free methods to work for inversion. The answer lays probably within the regularization/damping (Marquardt, 1970) which covers this structural gap. The important role of the first layer for Newton method has been highlighted recently by Wadia et al. (2021). In our case and in order to extend the inversion result to the first layer, we just introduce an additional layer before the input, which has the same size as the input, and does not bring any modification, hence the appellation "white layer". The white layer corresponds to a matrix multiplication with the identity, a null bias, an identity activation function and is unchanged during the training.

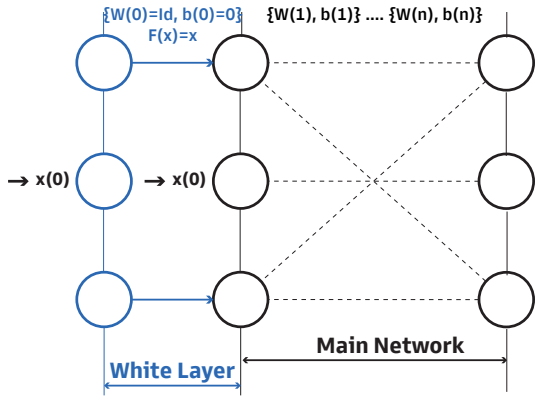

Figure 1: White layer extension before the input of the neural network.

Hypotheses relaxation: Some of the hypotheses of the main theorem could be relaxed. Notably the invertibility of the regularization could be limited to the last layer i.e. $\left\{\frac{\partial^2 \ell_R}{\partial x_p(n)^2}\right\}$. The inversion of $\left\{\frac{\partial^2 \ell_R}{\partial x_p(n)^2}\right\}$ is often trivial for e.g. it is the identity in the simplest case. Even in the worst-case scenario (e.g softmax function), the output layer is usually the smallest in the network. Inverting the matrix of second derivatives of the last layer (typically 10×10) is nowhere near the large size of the Hessian inversion (typically $10^6 \times 10^6$).

Admissible regularization: The ESN solution relies primarily on the existence of the term $\left(\frac{\partial \ell_R}{\partial x_p(k-1)}\right)$ which would be null if not for the admissible regularization we introduced early in this paper. The admissible regularization might seem atypical, however its effect revolves ultimately around minimizing the norm of the state variable i.e. $\|x\|_\Lambda$. Given that we are already minimizing the mismatches, and consequently the adjoints $\|b_p\|$ then we can assert that the admissible regularization is contributing to the gradient's norm minimization (Equ. 10), which is a desirable features for generalization purposes (Barrett & Dherin, 2020; Smith et al., 2021).

The Exact Stochastic Newton (ESN) solution provides scaling of the gradient and changes the bias update. Nevertheless, it remains unsuitable for direct implementation due to non-convexity. In the next section, we focus on correcting the shortcomings of ESN.

3 Adjustment of the Exact Stochastic Newton (ESN)

The ESN solution is exact and simple to compute; however, it diverges if implemented as it is. In this section, we first address the non-convexity issue and adjust the solution to become a descent direction. We also explore the stochasticity effect of the pattern randomness during the training process and provide an enhancement to the ESN. For the remainder of this paper we assume also that regularization is a of the same form as Equ. 4. We start this section by correcting the concave directions of the ESN.

3.1 Escaping the Saddles: a Descent Direction

The minimal rank solution (Equ. 17) has some similarities with the gradient expression (Equ. 10), yet it is not a descent direction. To further illustrate this, we consider the inner product of the Newton and the gradient expressions layer by layer. For the weights, the product is always positive:

$$\langle N(k), G(k)\rangle = \langle x_p(k-1), \Lambda_p(k-1)x_p(k-1)\rangle = \|x_p(k-1)\|^2_{\Lambda_p(k-1)} \geq 0. \tag{19}$$

However, most of the Newton biases corresponds to ascent directions :

$$\langle \eta(k), g(k)\rangle = -\|x_p(k-1)\|^2_{\Lambda_p(k-1)} + \mathbf{1}_{k=n}\|d_p - x_p(n)\|^2_2. \tag{20}$$

The last layer bias nature depend on the values of the regularization and the error. One of the main advantages of the Sifrian formulation is that we can treat the weights and biases separately and we can exactly pinpoint to the origin of non-convexity: the bias update. We choose to build our saddle-free variation by just changing the signs of the ascent direction. This manipulation is straightforward for all layers except the output bias update. The problematic term corresponds to the "granular" solution (i.e., no regularization), which by construction overfits to every pattern. We chose to keep the "granular" solution as is and switch the sign of the remaining term. This guarantees that the update is a descent direction[5]. The saddle-free ESN is the following:

$$\begin{cases} N(k) = -\frac{y_p(k)x_p(k-1)^T}{\|y_p(k)\|_2^2}\Lambda_p(k-1), \text{ with: } y_p(k) = \nabla F(a_p(k))\, b_p(k), \\ \\ \eta(k) = \mathbf{1}_{k=n}\left[\nabla F(a_p(n))\right]^{-1}(x_p(n) - d_p) + N(k)\, x_p(k-1). \end{cases} \tag{21}$$

The simplicity of the above convexity correction is unexpected and needs to be juxtaposed with the currently existing methods such as the Gauss Newton truncation (e.g. Schraudolph, 2002; Martens, 2010) or the absolute Hessian (Dauphin et al., 2014). The convexity correction is the first step in improving the basic ESN. Another intrinsic issue in the above formulation is that we try to optimize a given pattern without taking into consideration the yet "unknown" coming patterns. The essence of stochasticity in still missing. In the next section, we include such an aspect into the ESN method.

3.2 Randomization

The ESN solution we proposed so far is exact but remains deterministic since it focuses on training a single pre-selected pattern p. An intrinsic source of stochasticity in online learning stems from the randomness of the streamed patterns. As such, it is preferable to use random variables to describe the dynamics of ESN. The two fundamental equations of ESN after correcting convexity are:

$$\begin{cases} y_p(k)^T N(k) + \left(\frac{\partial \ell_R}{x_p(k-1)}\right)^T = 0, \\ \\ \nabla F(a_p(k))\left[-N(k)\, x_p(k-1) + \eta(k)\right] - \mathbf{1}_{k=n}\frac{\partial \ell_R}{x_p(n)} = 0. \end{cases} \tag{22}$$

Let's consider Ω as the set of possible outcomes[6] of the training patterns and define the following random variables:

$$\frac{\partial \ell_R}{x_p(k)} \to L(k,\Omega),\ \nabla F(a_p(k))\, b_p(k) \to Y(k,\Omega),\ x_p(k) \to X(k,\Omega),\ a_p(k) \to A(k,\Omega)). \tag{23}$$

The two main Sifrian equations could be rewritten after randomization as :

$$\begin{cases} Y(k,\Omega)^T N(k) + L(k-1,\Omega)^T = 0 \\ \\ \nabla F(A(k,\Omega))\,\eta(k) - \nabla F(A(k,\Omega))\, N(k)\, X(k-1,\Omega) - \mathbf{1}_{k=n} L(n,\Omega) = 0. \end{cases} \tag{24}$$

The least-squares version of the previous system is:

$$\begin{aligned} \text{argmin} \int d\Omega \left\| L(k-1,\Omega)^T + Y(k,\Omega)^T N(k) \right\|_F^2 \\ + \left\| \nabla F(A(k,\Omega))\left(\eta(k) - N(k)\, X(k-1,\Omega)\right) - \mathbf{1}_{k=n} L(n,\Omega)^T \right\|_2^2. \end{aligned} \tag{25}$$

We changed the ESN into a least-squares problem, which is desirable, given the sizeable well-established literature about optimal unbiased estimators for this class of problems (Aitken, 1936; Kariya & Kurata, 2004). Minimizing it, remains however convoluted and leads to terms that are difficult to estimate. As such, we will use a K-FAC like approximation (Martens & Grosse, 2015) and then carry the minimization.

[5]This point could be further justified with a first order Taylor development of the loss function.

[6]A $\sigma-$field is more appropriate, but we use only a random space Ω for the sake of simplicity.

Lemma 1 (K-FAC like approximation): If we assume the following (K-FAC like) approximation: $\forall k \,:\, \mathbb{E}\left[\nabla F\left(a_p(k)\right)^2 \otimes z\right] \simeq \mathbb{E}\left[\nabla F\left(a_p(k)\right)^2\right] \otimes \mathbb{E}\left[z\right]$ with z either $x_p\left(k-1\right)^T$ or $x_p\left(k-1\right)x_p\left(k-1\right)^T$ then the optimal least-squares solution involves a Sylvester equation :

$$\begin{cases} \mathbb{E}\left[y_{p,k}{y_{p,k}}^T\right]N_k + \mathbb{E}\left[\left(\nabla F_{p,k}\right)^2\right]N_k \text{Var}\left[x_{p,k-1}\right] = \\ \qquad\qquad -\mathbb{E}\left[y_{p,k}\left(\frac{\partial \ell}{\partial x_{p,k-1}}\right)^T\right] - \mathbf{1}_{k=n}\mathbb{E}\left[\nabla F_{p,k}\frac{\partial \ell}{\partial x_{p,n}}\left(x_{p,k-1} - \mathbb{E}\left[x_{p,k-1}\right]\right)^T\right], \\ \eta_k = N_k\mathbb{E}\left[x_{p,k-1}\right] + \mathbf{1}_{k=n}\mathbb{E}\left[\left(\nabla F_{p,k}\right)^2\right]^{-1}\mathbb{E}\left[\nabla F_{p,k}\left(\frac{\partial \ell}{\partial x_{p,n}}\right)\right]. \end{cases} \tag{26}$$

Proof The proof is based on finding critical points and it is included in the Appendix (A.4). □

We notice that the above system could considerably simplify if the state variables $x_p\left(k-1\right)$ is normalized i.e. null mean and identity variance. This aspect has been extensively explored in Deep Learning through batch, weight, layer and group normalization (Ioffe & Szegedy, 2015; Salimans & Kingma, 2016; Ba et al., 2016b; Wu & He, 2018). In our case normalization is necessary to circumvent the resolution of the Sylvester equation. Normalizing the state variable corresponds to the following transformation:

$$x_p\left(k-1\right) \rightarrow \left(\sqrt{\text{Var}\left[x_p\left(k-1\right)\right]}\right)^{-1}\left(x_p\left(k-1\right) - \mathbb{E}\left[x_p\left(k-1\right)\right]\right). \tag{27}$$

Adding normalization into randomized ESN is left for future works. At this stage, and given that the regularization Equ. 4 tends to minimize the norm of the state variable, we assume that the effect of the variance $\text{Var}\left[x_{p,k-1}\right]$ is negligible. As such, after adding a damping factor ϵI to the inverted matrix, we end up with the following formulas for the weights and biases update.

$$\begin{cases} N_k = -\mathbb{E}\left[\epsilon I + y_{p,k}y_{p,k}^T\right]^{-1}\left(\mathbb{E}\left[y_{p,k}\left(\frac{\partial \ell_R}{\partial x_{p,k-1}}\right)^T\right] - \mathbf{1}_{k=n}\mathbb{E}\left[\nabla F_{p,k}\frac{\partial \ell_R}{\partial x_{p,n}}x_{p,k-1}^T\right]\right), \\ \eta_k = N_k\mathbb{E}\left[x_{p,k-1}\right] + \mathbf{1}_{k=n}\mathbb{E}\left[\left(\nabla F_{p,k}\right)^2\right]^{-1}\mathbb{E}\left[\nabla F_{p,k}\left(\frac{\partial \ell_R}{\partial x_{p,n}}\right)\right]. \end{cases} \tag{28}$$

The above solution could also be derived if we split Equ. 25 and optimize successively for N and η separately. For further simplicity, the "granular" terms could be removed.

Expectations and estimators: We used the expectation $\mathbb{E}$ extensively in this paper to derive the theoretical ESN solution. However, estimators would replace expectation during implementation. The performance and the computational cost would be directly related to the estimators choice: rank one, moving average, or even batch estimators are all possible with different compromises.

Probability distribution/weights: It is common to use weights in the context of least-squares regression. For Deep Learning, we could increase the importance of patterns that are not well assimilated into the network, i.e., high errors. This is possible for e.g. by choosing probabilities/weights proportional to errors' norms: $w_p \sim \|x_p(n) - d_p\|_2^2$. The ESN regression equations become:

$$\begin{cases} N_k = -\mathbb{E}\left[\epsilon I + w_p y_{p,k}y_{p,k}^T\right]^{-1}\left(\mathbb{E}\left[w_p y_{p,k}\left(\frac{\partial \ell_R}{\partial x_{p,k-1}}\right)^T\right] - \mathbf{1}_{k=n}\frac{\mathbb{E}\left[\nabla F_{p,k}\frac{\partial \ell_R}{\partial x_{p,n}}x_{p,k-1}^T\right]}{\mathbb{E}[w_p]}\right), \\ \eta_k = N_k\frac{\mathbb{E}[x_{p,k-1}]}{\mathbb{E}[w_p]} + \mathbf{1}_{k=n}\mathbb{E}\left[w_p\left(\nabla F_{p,k}\right)^2\right]^{-1}\mathbb{E}\left[w_p\nabla F_{p,k}\left(\frac{\partial \ell_R}{\partial x_{p,n}}\right)\right]. \end{cases} \tag{29}$$

Notes on convergence: Several convergence results for the Newton method exist (e.g Bottou & Le Cun, 2005; Byrd et al., 2011; Meng et al., 2020). The ESN convergence fits well within the general framework developed by Sunehag et al. (2009), under some mild and non-restrictive conditions.

To batch or not to batch: After randomization, ESN could be used in batch mode. However, there are no guarantees that such a solution solves the "batch" Sifrian equations. Therefore, we focus only on using historical data in a similar way to adaptive methods (e.g Kingma & Ba, 2014).

4 RESULTS

We validate the ESN method for FNN, using a classification task on MNIST dataset (Lecun et al., 1998), and also an autoencoder training with the CURVES dataset (Hinton & Salakhutdinov, 2006). Leaky Relu and He initializer (He et al., 2016; Glorot & Bengio, 2010) are used for all the training. Further implementation details are available in the Appendix (A.5). We report the results hereafter:

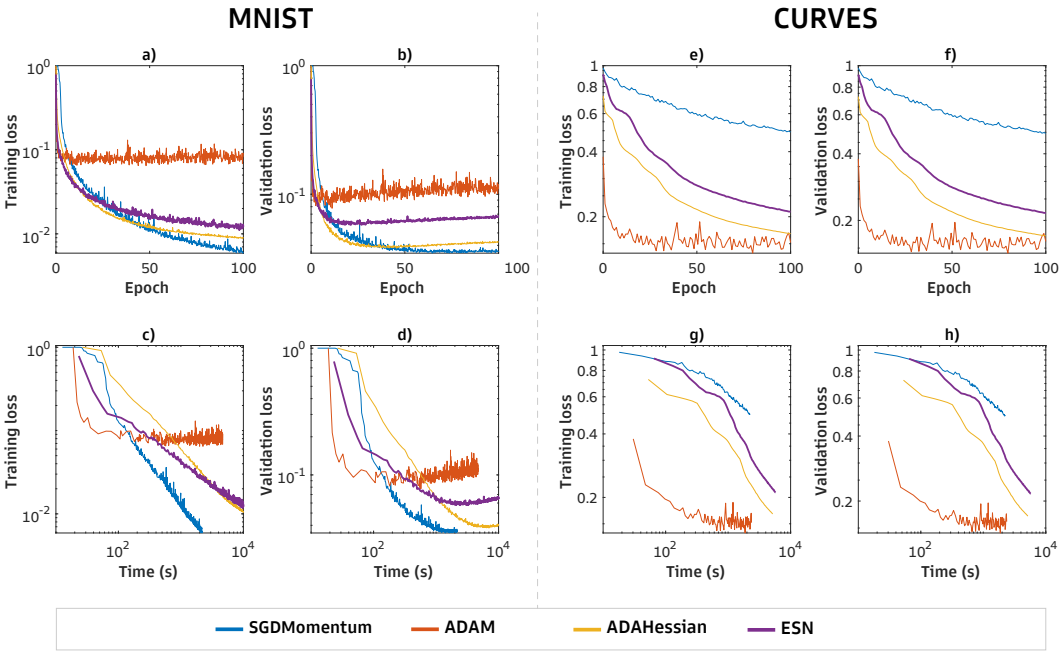

Figure 2: Training and validation loss as a function of both epochs and wall-clock time for classification (MNIST) and autoencoding (CURVES) with one seed. For MNIST classification the architecture is $\{784, 256, 32, 10\}$. For CURVES, the autoencoder has the following architecture $\{784, 216, 64, 6, 64, 216, 784\}$. Constant learning rates are used 0.001 for ADAM (Kingma & Ba, 2014) and ADAHessian (Yao et al., 2020) and 0.01 for SGD and ESN.

5 CONCLUSION

This paper showed that it is possible to derive an exact expression of the stochastic second-order update for Deep Learning via a second-order Lagrangian which we called Sifrian, under some regularization constraints. We revisited the basics of backpropagation and developed a new framework to better understand how does the Newton method operate. This framework yielded unexpected insights on how to correct the non-convexity, the importance of the first layer, and how to get a closed-form solution. We further proceeded to randomize the solution, to better align the training with its stochastic nature. We tested the solution using simple training models; and despite the known limitations of the Newton method, the ESN (Exact Stochastic Newton) solution was able to perform as well as adaptive methods in a non-convex setting. ESN holds great potential but still requires further refinement for batch training, generalization, parameters selection and estimators choice. Including normalization and generalization of this work to other type of networks architectures are other venues that need further investigation.

REPRODUCIBILITY STATEMENT

The results of this paper are reproducible. Commented Matlab code is included in the supplementary material. All experiments are run on a workstation with eight cores 2.6 GHz CPU.

REFERENCES

Naman Agarwal, Brian Bullins, and Elad Hazan. Second-order stochastic optimization for machine learning in linear time. *The Journal of Machine Learning Research*, 18(1):4148–4187, 2017.

Alexander C Aitken. Iv.—on least squares and linear combination of observations. *Proceedings of the Royal Society of Edinburgh*, 55:42–48, 1936.

Shun-Ichi Amari. Natural gradient works efficiently in learning. *Neural computation*, 10(2):251–276, 1998.

Shun-ichi Amari, Jimmy Ba, Roger Grosse, Xuechen Li, Atsushi Nitanda, Taiji Suzuki, Denny Wu, and Ji Xu. When does preconditioning help or hurt generalization? *arXiv preprint arXiv:2006.10732*, 2020.

Rohan Anil, Vineet Gupta, Tomer Koren, Kevin Regan, and Yoram Singer. Scalable second order optimization for deep learning. *arXiv preprint arXiv:2002.09018*, 2020.

Jimmy Ba, Roger Grosse, and James Martens. Distributed second-order optimization using kronecker-factored approximations. 2016a.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016b.

David GT Barrett and Benoit Dherin. Implicit gradient regularization. *arXiv preprint arXiv:2009.11162*, 2020.

Albert S Berahas, Majid Jahani, Peter Richtárik, and Martin Takáč. Quasi-newton methods for deep learning: Forget the past, just sample. *arXiv preprint arXiv:1901.09997*, 2019.

Raghu Bollapragada, Jorge Nocedal, Dheevatsa Mudigere, Hao-Jun Shi, and Ping Tak Peter Tang. A progressive batching l-bfgs method for machine learning. In *International Conference on Machine Learning*, pp. 620–629. PMLR, 2018.

Antoine Bordes, Léon Bottou, and Patrick Gallinari. Sgd-qn: Careful quasi-newton stochastic gradient descent. *Journal of Machine Learning Research*, 10:1737–1754, 2009.

Aleksandar Botev, Hippolyt Ritter, and David Barber. Practical gauss-newton optimisation for deep learning. In *International Conference on Machine Learning*, pp. 557–565. PMLR, 2017.

Léon Bottou and Yann Le Cun. On-line learning for very large data sets. *Applied stochastic models in business and industry*, 21(2):137–151, 2005.

Léon Bottou, Frank E Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *Siam Review*, 60(2):223–311, 2018.

Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

Charles George Broyden. The convergence of a class of double-rank minimization algorithms 1. general considerations. *IMA Journal of Applied Mathematics*, 6(1):76–90, 1970.

R Byrd, Gillian M Chin, Will Neveitt, and Jorge Nocedal. On the use of stochastic hessian information in unconstrained optimization. *SIAM Journal on Optimization*, 21(3):977–995, 2011.

Richard H Byrd, Samantha L Hansen, Jorge Nocedal, and Yoram Singer. A stochastic quasi-newton method for large-scale optimization. *SIAM Journal on Optimization*, 26(2):1008–1031, 2016.

Camille Castera, Jérôme Bolte, Cédric Févotte, and Edouard Pauwels. An inertial newton algorithm for deep learning. *Journal of Machine Learning Research*, 22(134):1–31, 2021.

Yann Dauphin, Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. *arXiv preprint arXiv:1406.2572*, 2014.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011.

Roger Fletcher. A new approach to variable metric algorithms. *The computer journal*, 13(3):317–322, 1970.

Thomas George, César Laurent, Xavier Bouthillier, Nicolas Ballas, and Pascal Vincent. Fast approximate natural gradient descent in a kronecker-factored eigenbasis. *arXiv preprint arXiv:1806.03884*, 2018.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256. JMLR Workshop and Conference Proceedings, 2010.

Donald Goldfarb. A family of variable-metric methods derived by variational means. *Mathematics of computation*, 24(109):23–26, 1970.

Donald Goldfarb, Yi Ren, and Achraf Bahamou. Practical quasi-newton methods for training deep neural networks. *arXiv preprint arXiv:2006.08877*, 2020.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pp. 448–456. PMLR, 2015.

Takeaki Kariya and Hiroshi Kurata. *Generalized least squares*. John Wiley & Sons, 2004.

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

Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. doi: 10.1109/5.726791.

Yann LeCun, D Touresky, G Hinton, and T Sejnowski. A theoretical framework for back-propagation. In *Proceedings of the 1988 connectionist models summer school*, volume 1, pp. 21–28, 1988.

Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pp. 9–48. Springer, 2012.

Dong C Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1):503–528, 1989.

Donald W Marquardt. Generalized inverses, ridge regression, biased linear estimation, and nonlinear estimation. *Technometrics*, 12(3):591–612, 1970.

James Martens. Deep learning via hessian-free optimization. In *ICML*, volume 27, pp. 735–742, 2010.

James Martens and Roger Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In *International conference on machine learning*, pp. 2408–2417. PMLR, 2015.

James Martens and Ilya Sutskever. Training deep and recurrent networks with hessian-free optimization. In *Neural networks: Tricks of the trade*, pp. 479–535. Springer, 2012.

Si Yi Meng, Sharan Vaswani, Issam Hadj Laradji, Mark Schmidt, and Simon Lacoste-Julien. Fast and furious convergence: Stochastic second order methods under interpolation. In *International Conference on Artificial Intelligence and Statistics*, pp. 1375–1386. PMLR, 2020.

Jorge Nocedal and Stephen Wright. *Numerical optimization*. Springer Science & Business Media, 2006.

Barak A Pearlmutter. Fast exact multiplication by the hessian. *Neural computation*, 6(1):147–160, 1994.

Sashank J Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond. *arXiv preprint arXiv:1904.09237*, 2019.

Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pp. 400–407, 1951.

Tim Salimans and Durk P Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *Advances in neural information processing systems*, 29:901–909, 2016.

Tom Schaul, Sixin Zhang, and Yann LeCun. No more pesky learning rates. In *International Conference on Machine Learning*, pp. 343–351. PMLR, 2013.

Nicol N Schraudolph. Fast curvature matrix-vector products for second-order gradient descent. *Neural computation*, 14(7):1723–1738, 2002.

Nicol N Schraudolph, Jin Yu, and Simon Günter. A stochastic quasi-newton method for online convex optimization. In *Artificial intelligence and statistics*, pp. 436–443. PMLR, 2007.

David F Shanno. Conditioning of quasi-newton methods for function minimization. *Mathematics of computation*, 24(111):647–656, 1970.

Samuel L Smith, Benoit Dherin, David GT Barrett, and Soham De. On the origin of implicit regularization in stochastic gradient descent. *arXiv preprint arXiv:2101.12176*, 2021.

Jascha Sohl-Dickstein, Ben Poole, and Surya Ganguli. Fast large-scale optimization by unifying stochastic gradient and quasi-newton methods. In *International Conference on Machine Learning*, pp. 604–612. PMLR, 2014.

Peter Sunehag, Jochen Trumpf, SVN Vishwanathan, and Nicol Schraudolph. Variable metric stochastic approximation theory. In *Artificial Intelligence and Statistics*, pp. 560–566. PMLR, 2009.

Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

Neha Wadia, Daniel Duckworth, Samuel S Schoenholz, Ethan Dyer, and Jascha Sohl-Dickstein. Whitening and second order optimization both make information in the dataset unusable during training, and can reduce or prevent generalization. In *International Conference on Machine Learning*, pp. 10617–10629. PMLR, 2021.

Yuxin Wu and Kaiming He. Group normalization. In *Proceedings of the European conference on computer vision (ECCV)*, pp. 3–19, 2018.

Zhewei Yao, Amir Gholami, Sheng Shen, Mustafa Mustafa, Kurt Keutzer, and Michael W Mahoney. Adahessian: An adaptive second order optimizer for machine learning. *arXiv preprint arXiv:2006.00719*, 2020.

A APPENDIX

A.1 $\mathcal{C}^2$ SIFRIAN EQUATIONS

The scalary product used in the Sifrian is either the inner product of vectors or the trace product for matrices, which are equivalent after vectorization:

$$\langle A, B\rangle = \text{trace}\left(B^T A\right) = \text{vec}\left(B\right)^T \text{vec}\left(A\right),$$

$$\langle x, y\rangle = x^T y = \text{trace}\left(xy^T\right).$$

The derivation is carried through perturbation factorization for e.g.:

$$\mathcal{S}\left(W\left(k\right) + \delta W\left(k\right)\right) - \mathcal{S}\left(W\left(k\right)\right) \sim \left\langle \frac{\partial \mathcal{S}}{\partial W\left(k\right)}, \delta W\left(k\right)\right\rangle.$$

We carry the derivation and we report hereafter the full Sifrian equations for a $\mathcal{C}^2$ activation functions:

$$\begin{cases}
(\mathbf{1})\ \sum_p -\nabla F\left(a_p(k)\right)\gamma_p\left(k\right)x_p(k-1)^T - \mathbf{1}_{k=2..n}\nabla F\left(a_p(k)\right)b_p(k)\zeta_p(k-1)^T \\
+\sum_p \nabla^2 F\left(a_p(k)\right)b_p(k)\left[N\left(k\right)x_p(k-1) + \eta\left(k\right) - \mathbf{1}_{k=2..n}W(k)\zeta_p(k-1)\right]x_p(k-1)^T \\
= \sum_p \nabla F\left(a_p(k)\right)b_p(k)x_p(k-1)^T, \\
(\mathbf{2})\ \sum_p \nabla^2 F\left(a_p(k)\right)b_p(k)\left[N\left(k\right)x_p(k-1) + \eta\left(k\right) - \mathbf{1}_{k=2..n}W(k)\zeta_p(k-1)\right] \\
+\sum_p -\nabla F\left(a_p(k)\right)\gamma_p\left(k\right) = \sum_p \nabla F\left(a_p(k)\right)b_p(k), \\
(\mathbf{3})\ \gamma_p\left(k\right) - \mathbf{1}_{k=1..n-1}W(k+1)^T\nabla F\left(a_p(k+1)\right)\gamma_p\left(k+1\right) + \mathbf{1}_{k=n}\zeta_p(n) + \lambda\mathbf{1}_{k=1..n-1}\zeta_p(k) \\
+\mathbf{1}_{k=1..n-1}W_{k+1}^T\nabla^2 F\left(a_p(k+1)\right)b_p(k+1)\left[N_{k+1}x_p(k) + \eta_{k+1} - \mathbf{1}_{k=1..n-1}W_{k+1}\zeta_p(k)\right] \\
+\mathbf{1}_{k=1..n-1}N\left(k+1\right)^T\nabla F\left(a_p(k+1)\right)b_p(k+1) = 0, \\
(\mathbf{4})\ \zeta_p(k) - \mathbf{1}_{k=2..n}\nabla F\left(a_p(k)\right)W(k)\zeta_p(k-1) + \nabla F\left(a_p(k)\right)\left[N\left(k\right)x_p(k-1) + \eta\left(k\right)\right] = 0.
\end{cases}$$

A.2 ESN RESOLUTION FOR $\mathcal{C}^2$ ACTIVATION FUNCTIONS

The stochastic case for $\mathcal{C}^2$ functions has the same type of simplifications. Comparing the first and second equations:

$$\begin{aligned}
&(\mathbf{1})\ -\nabla F\left(a_p(k)\right)\gamma_p\left(k\right)x_p(k-1)^T - \mathbf{1}_{k=2..n}\nabla F\left(a_p(k)\right)b_p(k)\zeta_p(k-1)^T \\
&+\nabla^2 F\left(a_p(k)\right)b_p(k)\left[N\left(k\right)x_p(k-1) + \eta\left(k\right) - \mathbf{1}_{k=2..n}W(k)\zeta_p(k-1)\right]x_p(k-1)^T \\
&= \nabla F\left(a_p(k)\right)b_p(k)x_p(k-1)^T, \\
&(\mathbf{2})\ \nabla^2 F_{p,k}b_p(k)\left[N_k x_p(k-1) + \eta_k - \mathbf{1}_{k=2..n}W_k\zeta_p(k-1)\right] - \nabla F_{p,k}\gamma_p\left(k\right) = \nabla F_{p,k}b_p(k).
\end{aligned}$$

This yields that $\mathbf{1}_{k=1..n-1}\zeta_p(k) = 0$. The fourth equation state that $\left[N\left(k\right)x_p(k-1) + \eta\left(k\right)\right] = 0$ is always null except for the last layer. Let's focus on the last layer. Three equations allows to characterize it:

$$\begin{cases}
(\mathbf{2})\ b_p(n) = \nabla F_{p,n}^{-1}\nabla^2 F_{p,n}b_p(n)\left[N\left(n\right)x_p(n-1) + \eta\left(n\right)\right] - \gamma_p\left(n\right), \\
(\mathbf{3})\ \gamma_p\left(n\right) + \zeta_p(n) = 0, \\
(\mathbf{4})\ \zeta_p(n) + \nabla F\left(a_p(n)\right)\left[N\left(n\right)x_p(n-1) + \eta\left(n\right)\right] = 0.
\end{cases}$$

If we assume that the $\mathcal{C}^2$ has a non null second derivative everywhere then, the previous three equations yield a specific value of $[N(n)\, x_p(n-1) + \eta(n)]$ and also $\zeta_p(n)$:

Except the last layer we have: $\gamma_p(k) = -b_p(k)$, and a similar system to the piecewise affine case could be derived.

A.3 EXAMPLE: A SIMPLE REGULARIZATION

In this section we give a concrete simple example to further explain the ESN method. In the case of an l^2 cost function and regularization defined by:

$$R(x) = \frac{\lambda}{2} \sum_{p \in D} \sum_{k=1..n-1} \langle x_p(k), x_p(k) \rangle$$

we have the following solution:

$$\begin{cases} N(k) = -\lambda \frac{\nabla F(a_p(k)) b_p(k) x_p(k-1)^T}{\|\nabla F(a_p(k)) b_p(k)\|_2^2}, \\ \\ \eta(k) = \nabla \left[F(a_p(k))\right]^{-1} (d_p - x_p(n))\, \mathbf{1}_{k=n} - N(k)\, x_p(k-1). \end{cases}$$

- The previous solution is straightforward to compute; the first order backpropagation provides all the elements.
- It is co-linear with the gradient except for the bias of the last layer.
- The adjoints $b_p(k)$ are impacted by choice of the regularization parameter λ.
- The minimal rank Newton solution is not a descent direction. Most of the biases correspond to an ascent direction.
- The squared norm division is problematic and will lead to divergent solution if the norm of the backpropagated errors is very small i.e. $\|\nabla F(a_p(k))\, b_p(k)\| \to 0$. A sufficient condition to avoid divergence consists in defining layer by layer regularization such as: $\lambda_{k-1} \propto \|\nabla F(a_p(k))\, b_p(k)\|$.
- It is possible to remove the regularization i.e. $\lambda \to 0$, this yields a "granular" solution which states that the second order optimal solution consists in just adjusting the last layer bias with a slope corrected error: $\nabla \left[F(a_p(k))\right]^{-1} (d_p - x_p(n))$. The granular solution is not suitable for neural network training, as it overfits each pattern.

A.4 PROOF: PROPOSITION RANDOMIZED ESN

We start from the randomized least-squares criteria.

$$\begin{aligned} \text{argmin} \int d\Omega \left\| L(k-1,\Omega)^T + Y(k,\Omega)^T N(k) \right\|_F^2 \\ + \left\| \nabla F(A(k,\Omega)) \left(\eta(k) - N(k)\, X(k-1,\Omega)\right) - \mathbf{1}_{k=n} L(n,\Omega)^T \right\|_2^2. \end{aligned}$$

The optimum is a critical point i.e. :

$$\forall k \left(\frac{\partial}{\partial N(k)}, \frac{\partial}{\partial \eta(k)} \right) = (0,0).$$

Each layer could be optimized separately which is desirable feature that reduce the dimensionality of the problem. We get the following derivatives:

$$\begin{aligned} \frac{\partial}{\partial N(k)} \to \int d\Omega\, Y(k)\, Y(k)^T N(k) + Y(k)\, L(k-1,\Omega)^T - \nabla F(A(k,\Omega)) \\ \left(\nabla F(A(k,\Omega)) \left(\eta(k) - N(k)\, X(k-1,\Omega)\right) - \mathbf{1}_{k=n} L(n,\Omega)\right) X(k-1,\Omega)^T = 0, \end{aligned}$$

$$\frac{\partial}{\partial \eta(k)} \to \int \nabla F(A(k,\Omega))^2 \left(\eta(k) - N(k)\, [X(k-1)]\right) - \nabla F(A(k,\Omega))\, \mathbf{1}_{k=n} \left[L(n)^T\right] = 0.$$

Integration over Ω yields expectations. Using the original notations we get:

$$\frac{\partial}{\partial N(k)} \to \mathbb{E}\left[y_p(k)\, y_p(k)^T\right] N(k) + \mathbb{E}\left[y_p(k)\left(\frac{\partial \ell}{\partial x_p(k-1)}\right)^T\right] - \mathbb{E}\left[\nabla F(a_p(k))^2 \left(\eta(k)\, x_p(k-1)^T - N(k)\, x_p(k-1)\, x_p(k-1)^T\right)\right] + \mathbf{1}_{k=n}\mathbb{E}\left[\nabla F(a_p(k))\frac{\partial \ell}{\partial x_p(n)} x_p(k-1)^T\right] = 0$$

$$\frac{\partial}{\partial \eta(k)} \to \mathbb{E}\left[\nabla F(a_p(k))^2 \left(\eta(k) - N(k)\, x_p(k-1)\right)\right] - \mathbf{1}_{k=n}\mathbb{E}\left[\nabla F(a_p(k))\frac{\partial \ell}{\partial x_p(n)}\right] = 0.$$

We apply the K-FAC (like) approximation and we get:

$$\frac{\partial}{\partial N(k)} \to \mathbb{E}\left[y_p(k)\, y_p(k)^T\right] N(k) + \mathbb{E}\left[y_p(k)\left(\frac{\partial \ell}{\partial x_p(k-1)}\right)^T\right] - \mathbb{E}\left[\nabla F(a_p(k))^2\right]\left(\eta(k)\,\mathbb{E}\left[x_p(k-1)^T\right] - N(k)\,\mathbb{E}\left[x_p(k-1)\, x_p(k-1)^T\right]\right) + \mathbf{1}_{k=n}\mathbb{E}\left[\nabla F(a_p(k))\frac{\partial \ell}{\partial x_p(n)} x_p(k-1)^T\right] = 0.$$

$$\frac{\partial}{\partial \eta(k)} \to \mathbb{E}\left[\nabla F(a_p(k))^2\right]\left(\eta(k) - N(k)\,\mathbb{E}\left[x_p(k-1)\right]\right) + \mathbf{1}_{k=n}\mathbb{E}\left[\nabla F(a_p(k))\frac{\partial \ell}{\partial x_p(n)}\right] = 0.$$

We further simplify the system by right multiplying the last equation with $\mathbb{E}\left[x_p(k-1)\right]$ and summing:

$$\frac{\partial}{\partial N(k)} + \frac{\partial}{\partial \eta(k)}\mathbb{E}\left[x_p(k-1)\right] \to \mathbb{E}\left[y_p(k)\, y_p(k)^T\right] N(k) + \mathbb{E}\left[y_p(k)\left(\frac{\partial \ell}{\partial x_p(k-1)}\right)^T\right] + \mathbb{E}\left[\nabla F(a_p(k))^2\right]\left(N(k)\,\mathrm{Var}\left[x_p(k-1)\right]\right) + \mathbf{1}_{k=n}\mathbb{E}\left[\nabla F(a_p(k))\frac{\partial \ell}{\partial x_p(n)}\left(x_p(k-1) - \mathbb{E}\left[x_p(k-1)\right]\right)^T\right] = 0.$$

The above expression corresponds to a Sylvester equation. The last two equations are the main result of the proposition.

A.5 Implementation details:

In this section we specify technical details of our implementation. First, we remove the "granular" terms from our solution, since they do overfit each pattern by construction. The solution, after adding a damping factor ϵI using the original variables, is:

$$\begin{cases} N(k) = -\left(\mathbb{E}\left[\epsilon I + y_p(k)\, y_p(k)^T\right]\right)^{-1}\mathbb{E}\left[y_p(k)\left(\frac{\partial \ell_R}{x_p(k-1)}\right)^T\right], \\ \eta(k) = N(k)\,\mathbb{E}\left[x_p(k-1)\right]. \end{cases} \tag{30}$$

We used $y_p(k) = \nabla F(a_p(k))\, b_p(k)$ to keep the equations compact.
ESN is solved for each layer independently, which means that the inverted matrices are a scale

smaller than the empirical Fisher matrix for e.g.. To further illustrate this point, let's consider, for e.g., the feedforward network with layers sizes $[784, 256, 32, 10]$ (MNIST classification): the empirical Fisher matrix would have a size of 210298x210298, while the biggest matrix inversion for ESN would be only 256x256. In practice, the computation requires two expectation $\left(\mathbb{E}\left[y_p\left(k\right)\left(\frac{\partial \ell_R}{x_p(k-1)}\right)^T\right], \mathbb{E}\left[x_p\left(k-1\right)\right]\right)$ which could be estimated using exponentially decaying moving averages (Similar to ADAM). The product with the non-centered precision matrix $\left(\mathbb{E}\left[\epsilon I + y_p\left(k\right) y_p\left(k\right)^T\right]\right)^{-1}$, could be approximated in several ways. We list hereafter at least three methods:

- Direct method: the non-centered covariance matrix could be estimated using an exponentially decaying moving average and inverted directly given the small sizes.
- Recursive Least-Squares (RLS): A sliding window could be used to update recursively the estimation of $N(k)$ using Sherman-Morrison formula with an initialization of the precision matrix at $\frac{I}{\epsilon}$. Such a method is a classic of online learning and is similar to Kalman filtering.
- Rank one method: instead of involving the historical data, the simplest/crudest estimation of the covariance matrix is based on the spot/instantaneous information. The precision matrix could be derived using Sherman-Morrison formula (applied once):

$$\left(\epsilon I + y_p\left(k\right) y_p\left(k\right)^T\right)^{-1} = \frac{1}{\epsilon}\left(I - \frac{y_p\left(k\right) y_p\left(k\right)^T}{\epsilon + \left\|y_p\left(k\right)\right\|_2^2}\right) \tag{31}$$

  The denominator from the previous formula suggests the following choice of adaptive damping $\epsilon \sim \left\|y_p\left(k\right)\right\|_2^2$.

After all the previous considerations, and given the novelty of the method, the candidate solution we pick here for experimentation is:

$$\begin{cases} N\left(k\right) = -\mathbb{M}_\kappa\left[\epsilon I + w_p y_{p,k} y_{p,k}^T\right]^{-1} \mathbb{M}_\kappa\left[w_p y_p\left(k\right)\left(\frac{\partial \ell_R}{x_p(k-1)}\right)^T\right], \\ \\ \eta\left(k\right) = N\left(k\right) \frac{\mathbb{M}_\kappa[w_p(k) x_p(k-1)]}{\mathbb{M}_\kappa[w_p]}. \end{cases} \tag{32}$$

The operator $\mathbb{M}_\kappa$ is the exponential moving average with a forgetting factor κ, which has typical values between $0.9 \sim 0.99$. The forgetting factor is often referred to as β (e.g., in ADAM), but we had to choose a different letter since we had already used β for the biases.
The results are reported in the main text