A Generalized Learning Approach to Deep Neural Networks

Francesca Ponti[1], Fabrizio Frezza[1], Patrizio Simeoni[2], and Raffaele Parisi[1]

[1] *"Sapienza" University of Rome, Rome, Italy,*
[2] *South East Technological University, Carlow, Ireland*

https://doi.org/10.26636/jtit.2024.3.1454

Abstract — Optimization of machine learning architectures is essential in determining the efficacy and the applicability of any neural architecture to real world problems. In this work a generalized Newton's method (GNM) is presented as a powerful approach to learning in deep neural networks (DNN). This technique was compared to two popular approaches, namely the stochastic gradient descent (SGD) and the Adam algorithm, in two popular classification tasks. The performance of the proposed approach confirmed it as an attractive alternative to state-of-the-art first order solutions.
Due to the good results presented in the case of shallow DNN, in the last part of the article an hybrid optimization method is presented. This method consists in combining two optimization algorithms, i.e. GNM and Adam or GNM and SGD, during the training phase within the layers of the neural network. This configuration aims to benefit from the strengths of both first- and second-order algorithms. In this case a convolutional neural network is considered and its parameters are updated with a different optimization algorithm. Also in this case, the hybrid approach returns the best performance with respect to the first order algorithms.

Keywords — deep neural networks, machine learning, optimization

1. Introduction

In recent years, neural deep learning (NDL) [1]–[5] has been acquiring popularity in many applicative fields of engineering, such as system identification and control, robotics, communication systems and signal processing [6]. As a matter of fact, deep neural networks (DNNs) are able to handle large amounts of data that would be hard to treat by more traditional techniques.

A neural network (NN) is generally composed by a set of parameters (the *weights*) that are updated iteratively in order to establish the desired correspondence between the input and the output. This phase is called *training* and it is performed by means of properly selected optimization algorithms.

In general, optimization consists in the minimization or maximization of a function subject to specific constraints on its variables [7]. In the NN context, optimization corresponds to finding the optimal weights $\mathrm{bf}w^*$ that minimize a given function $E(\mathrm{bf}w)$ (the *loss function*):

$$\mathrm{bf}w^* = \operatorname*{argmin}_{\mathrm{bf}w} E(\mathrm{bf}w) \; . \tag{1}$$

Several choices for the objective function $E(\mathrm{bf}w)$ are possible. Due to the high degree of non-linearity of $E(\mathrm{bf}w)$, minimization is performed by employing iterative techniques and it is not always guaranteed that the obtained minima are optima (*local minima*).

Optimization methods can be roughly grouped in two main classes [8], [9]: first and second-order methods.

First order methods are the most commonly applied in practice. They involve the use of the first derivative of the objective function with respect to the weights in order to choose the direction of movement in the search space.

First-order methods are quite popular in machine learning, since they have a low computational cost and are usually easy to implement. However, as highlighted in [10], first-order methods have several shortcomings, since they depend on hyper-parameter selection, they are extremely hard to tune and parallel computing opportunities are often limited.

In order to find a more effective alternative to first-order methods and to overcome their limitations, second-order methods [7] have been often employed in traditional shallow neural networks. They resulted to work well in several machine learning tasks [10]–[13]. The standard second-order technique is the *Newton's method*, that makes use of the Hessian matrix, formed by the second-order derivatives of the error functional, and has a quadratic rate of convergence. However, the Newton's method requires exact computation of the Hessian matrix, which is computationally expensive.

For this reason quasi-Newton's methods have been introduced [14]. They are based on different approximations of the inverse of Hessian matrix, with the aim of combining the rate of convergence of Newton's method with the scalability typical of first-order methods.

Moreover, another issue that affects machine learning, and in particular deep learning algorithms, are saddle points, which lead to reaching non-optimal minimum points.

Recently, many studies are proving that in many cases second order methods are more efficient than first order methods at avoiding saddle points [15], [16] reversing the traditional hypothesis that proposes as more performing the first order methods [17].

In this work we propose a general learning framework that belongs to the class of quasi-Newton's methods. This approach was introduced as a powerful learning algorithm for feedforward and recurrent NNs [18] and it was shown to belong to the class of generalized Newton's methods (GNMs).

We introduce its application to DNNs and compare it to most common optimization techniques in two deep-learning benchmark classification tasks. Moreover, due to the good performances achieved for the classification of a deep shallow neural network, an hybrid optimization approach is proposed to train a convolutional neural network (CNN). This method consists in combining two optimization algorithms, i.e. GNM and Adam or GNM and SGD, during the training phase within the layers of the neural network.

The proposed hybrid optimization approach aims to combine the speed of convergence of the second order optimization method under examination, i.e. GNM, with the low computational complexity of first order method that is required to train convolutional layers.

Indeed, as already stated, the algorithm proposed was tested in a feed-forward neural networks in the past years, when deep learning had not yet taken hold.

Then, testing the performance of existing optimization algorithms that have not been previously used on deep neural networks brings novelty by exploring unexplored territory, potentially improving performance, enabling comparisons against state-of-the-art methods, broadening applicability, and driving methodological advancements.

This paper contains a review of SGD and Adam methods in Section 2. In the same section, the description of the generalized Newton method under examination is reported. The experimental results obtained in two classification tasks, that consider MNIST and SUSY dataset, are described in Section 3. Section 4 describes the hybrid approach proposed and shows its result in case of a CNN. The conclusions are reported in Section 5.

2. Existing Optimization Algorithms

2.1. Stochastic Gradient Descent

Gradient descent (GD) [19] is a very popular iterative first-order method. At the generic k-th iteration it is represented by the following equation:

$$\mathrm{bf}w_{k+1} = \mathrm{bf}w_k - \eta\,\nabla_{\mathrm{bf}w_k} E \ , \tag{2}$$

where the *gradient* of the error functional with respect to the weights $\nabla_{\mathrm{bf}w_k} E$ is employed to compute the movement *in the weight space*. η (the *step length* or *learning rate*) is usually selected empirically and determines the rate of convergence of the loss function. This parameter represents how much the innovation term $\nabla_{\mathrm{bf}w} E$ influences the new weights with respect to the old ones.

The stochastic gradient descent (SGD) [20] is a modification of the GD based on an approximation of the actual gradient. Application of the SGD to NNs led to the well-known back-propagaton algorithm [21], [22]. One of the main advantages of SGD is its computational efficiency. It processes training data in small batches, making it well-suited for large datasets. SGD also has a low memory footprint since it only requires storing a small batch of data at a time. Additionally, SGD can converge faster compared to batch gradient descent as it quickly updates the model parameters using each batch of data.

However, SGD has some limitations that need to be considered. One major drawback is its susceptibility to getting stuck in local minima due to the high variance in the stochastic gradients. This can lead to suboptimal solutions.

SGD also requires careful tuning of the learning rate, as a too high learning rate can cause divergence and a too low learning rate can result in slow convergence. Additionally, SGD may struggle with handling noisy or sparse gradients, requiring the use of additional techniques such as learning rate schedules or momentum.

2.2. Adam Optimizer

One of the most popular first-order optimization algorithms in DNNs is the Adam optimizer [23]. Adam is an adaptive learning rate method, in the sense that it adapts the learning rate for different parameters by computing the first and second moments of the weight gradients.

The Adam weight update formula is expressed as follows:

$$w_k = w_{k-1} - \eta \frac{\hat{m}_{k+1}}{\sqrt{\hat{v}_k} + \epsilon} \tag{3}$$

where w_k is the weight at the k-th iteration and $\hat{m}_k$ and $\hat{v}_k$ the bias corrected estimators of the moving averages.

The advantages of the Adam optimizer lie in its adaptive learning rate, efficiency in handling sparse gradients, momentum optimization, robustness to hyperparameters, and wide applicability. These features contribute to its effectiveness and popularity in the field of deep learning.

However, there are some drawbacks to consider. Adam requires additional memory and computation resources due to its storage and update of past gradients and squared gradients. It can be sensitive to the choice of learning rate, and setting it improperly may lead to convergence issues.

2.3. Generalized Newton's Method

Here we briefly recall the main concepts related with the GNM presented in [18]. As stated in [18], a feedforward NN is a sequence of layers, each one composed by a linear and non-linear block [2], [4], [5] (see Fig. 1).

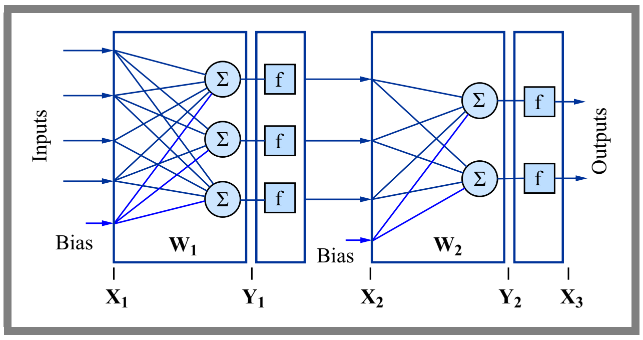

Fig. 1. Neural network representation.

At the k-th iteration of the learning procedure, in the forward propagation phase the output of the linear block of the generic n-th layer ($n = 1, \dots, L$) is computed with the following equation:

$$\mathbf{Y}_k^n = \mathbf{X}_k^n \mathbf{W}_k^n \tag{4}$$

In Eq. (4) each row of matrices $\mathbf{X}_k^n$ and $\mathbf{Y}_k^n$ refers to the generic learning pattern, while the generic column of matrix $\mathbf{W}_k^n$ contains the weights leading to the generic output neuron.

The non-linear block computes the output of the non-linear part, that is the input to the next $(n+1)$-th layer:

$$\mathbf{X}_k^{n+1} = [f\{\mathbf{Y}_k^n\} \ \mathbf{1}], \tag{5}$$

where f represents the activation function and **1** is the column of the bias inputs. $\mathbf{X}_k^{L+1}$ is the *global* output of the network.

Considering the generic n-th layer ($n = 1, \dots, L$) of the NN, the GNM employs the following formula to compute the new weights at the next $(k+1)$-th iteration:

$$\mathbf{W}_{k+1}^n = \mathbf{W}_k^n - \eta(\mathbf{X}_k^n)^+ \nabla_{\mathbf{Y}_k^n} E, \tag{6}$$

where $(\mathbf{X}_k^n)^+$ is the *pseudoinverse* of matrix $\mathbf{X}_k^n$.

Introducing the estimated correlation matrix of the inputs to the n-th layer $\Phi_k^n = (X_k^n)^T X_k^n$ and relating the *gradient matrix in the neuron space* $\nabla_{\mathbf{Y}_k^n} E$ with the *gradient matrix in the weight space* $\nabla_{\mathbf{W}_k^n} E$ it can be shown that:

$$\mathbf{W}_{k+1}^n = \mathbf{W}_k^n - \eta(\mathbf{\Phi}_k^n)^+ \nabla_{\mathbf{W}_k^n} E. \tag{7}$$

Equation (7) represents a *modified Newton's method* applied to the generic n-th layer. In [18] it is shown that, extending the previous formula to the whole network and comparing to the Newton's formula, the local Hessian matrix H_k (that is the Hessian related with the single layer) is approximated by $diag(\Phi_k)$. Since Φ_k is a diagonal matrix, this approximation assumes that the second-order partial derivatives with respect to the weights belonging to different layers are zero, that is neurons with respect to different layers are decoupled.

Moreover, it is well known that the Hessian matrix may not satisfy the property of positive definiteness. In contrast, the matrix Φ_k satisfies always the property of semipositive definiteness and almost always the property of positive definiteness.

This feature is important as it ensures that the algorithm assumes a descent direction. Indeed, it represents one of the main advantages of the considered algorithm. However, as it will be described in Section 4, the weights update Eq. (6) depends on the pseudoinverse of the input vector of each layer of the neural network. This type of operation could be computationally expensive in case of convolutional neural networks, in which tensors are involved.

This limitation is overcomed by the proposal of a hybrid optimization approach that combines the benefits and potential of both first and second-order methods to achieve higher performance.

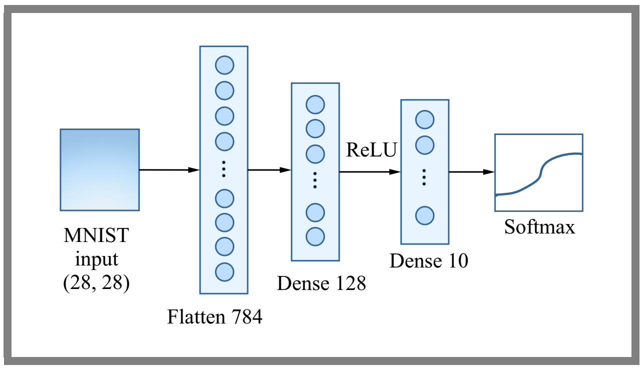

Fig. 2. Neural network architecture for MNIST classification.

3. Experimental Results

In this section we describe the results obtained in two typical experimental tasks considering two shallow NNs.

3.1. MNIST Dataset Classification Task

The first classification task used the MNIST dataset [24]. MNIST is a dataset of 60 000 squared 28×28 pixel grayscale images of handwritten single digits between 0 and 9 [25]. The aim of the network is to classify an image of a handwritten digit into one of 10 classes corresponding to the integer values from 0 to 9. A neural network with three layers was considered (Fig. 2).

The first layer in this network, called *flatten layer*, transforms the images from a two-dimensional array (28×28 pixels) to a one-dimensional array (28×28 = 784 pixels). This layer has no parameters to learn, it only reshapes the data.

In cascade with the flatten layer, the network is made of a sequence of two dense or *fully connected* layers. The first dense layer has 128 nodes (or neurons), with activation function *ReLU*. The last layer returns a logit array of length 10 and uses the activation function *softmax*. Each node of the last layer returns the probability that the current image belongs to one of the 10 classes.

The problem taken into account is an example of *multi-class classification task*. In order to evaluate the performance, the categorical cross entropy loss function has been considered:

$$E = -\sum_{i=1}^{10} y_i log(\hat{y}_i), \tag{8}$$

where $\hat{y}_i$ is s the i-th output of the network while y_i is the corresponding target value.

The generalized Newton's method (GNM) [18] was compared to the SGD and the Adam optimizer. The training process was iterated 20 times, considering a batch size of different lengths. Figure 3 shows the results obtained with batches of length 32, 64, and 128 in the case of the GNM optimizer. The batch size of 32 reached the lowest loss value and was selected in all subsequent tests. Same results were obtained with the SGD [19], [20] and Adam [23] optimizers.

In all cases weights were initialized by using a random uniform distribution between –1 and 1. The learning rate η was empirically selected. In particular, η was set to 0.001 for the

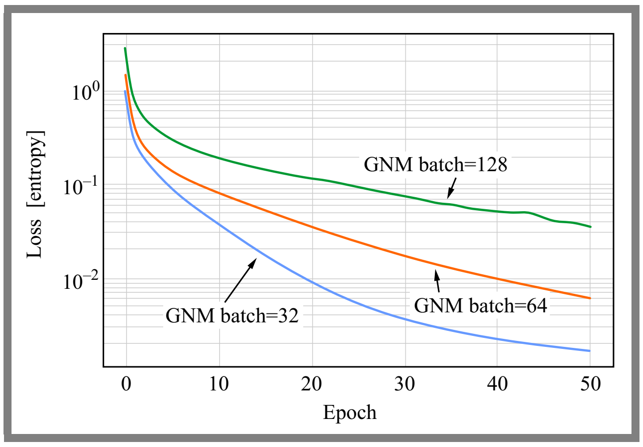

Fig. 3. MNIST dataset: GNM loss function comparison with respect to batch size.

Adam and SGD algorithms, while for the GNM approach $\eta = 16$ was considered. All simulations were carried out using the TensorFlow library [26]. Of course, since the GNM optimizer is not included in the default library, it has been defined and integrated into the eager execution. In eager execution TensorFlow does not build graphs, and each operation is executed by Python immediately. This feature makes it adapt to research and experimentation.

Conversely, the graph execution builds graph as representation of tensor computation and evaluates the model. The selection of the number of epochs, for the current case and all the other cases under examination, was established basing on two main factors: the convergence, and computational speed. In particular, the number of epochs are selected based on observing when the loss performance metric stabilizes in terms of convergence, with the goal to ensure that the model has converged and reached its optimal performance.

Figure 4 shows the loss function as a function of the number of epochs in a semilogaritmic scale.

As expected, the GNM loss curve has a faster convergence with respect to the first order optimizers, i.e. Adam and SGD.

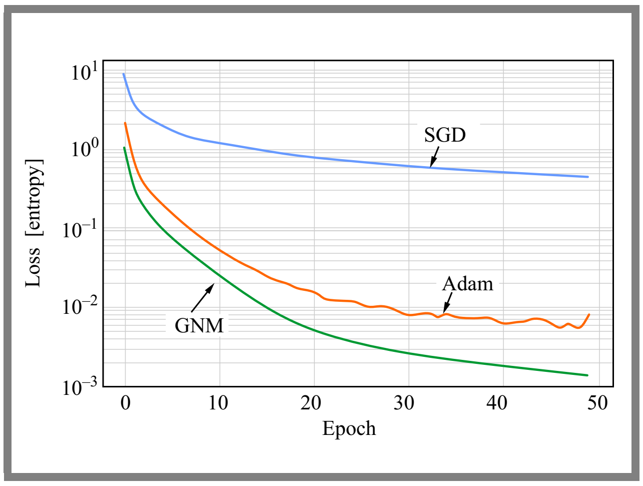

Fig. 4. MNIST dataset: GNM, SGD, and Adam loss function comparison with respect to epochs in semilogarithmic scale.

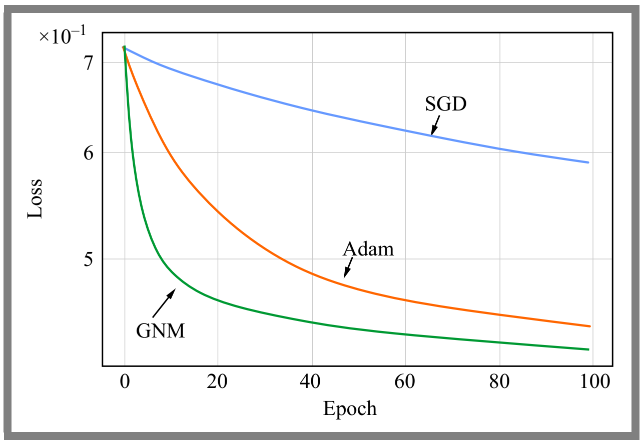

Fig. 5. SUSY dataset: GNM, SGD, and Adam loss function comparison with respect to epochs in semilogarithmic scale.

3.2. SUSY Dataset Classification Task

The second classification task is based on the SUSY dataset [27] and consists in recognizing the presence of supersymmetric particles in collisions at high energy colliders.

More specifically, the problem is a binary classification task where the goal is to establish whether each data point, generated via Monte Carlo simulations and characterized by 18 features (or kinematic variables), represents a signal *potential collision*, labeled with 1, or just *background*, labeled with 0. The first eight features are direct measurements of the kinematic features of final state particles in the accelerator.

The other ten features are high-level features derived from the first eight features and help in discriminating between the two classes. The whole dataset consists of 5 million points, of which 60 000 were considered as training set.

In this work, we propose a deep-learning architecture with the target of estimating the probability that an event is from a signal or a background process. The same approach was considered in [27]. The architecture considered is a neural network with four layers, having 256 units in the first layer, 128 in the second, 32 in the third layer, and one single unit in the last layer. This last layer uses a sigmoidal activation function while the other layers use the hyperbolic tangent function.

For each learning algorithm, the learning rate was selected by considering the best performance in terms of learning rate and weight initialization. Also in this experiment, weights were initialized by a random uniform distribution in the range [–1, 1]. The learning rate selected for the Adam optimizer was 0.001, for the gradient descent was 0.1, and 500 000 for the generalized Newton method.

In order to justify this last choice, from Eq. (6) we observe that in this case the learning rate depends on the condition number of the input matrix, i.e. on the order of magnitude of the inputs.

Figure 5 shows the average loss function of the described algorithms with respect to the epochs.

Also in this case, the GNM loss function was characterized by higher rate of convergence with respect to the Adam and SGD optimizers.

4. Hybrid Optimization Approach

In this section the results obtained with an hybrid optimization approach are reported.

As described in the previous chapter, the GNM has shown the best performances with respect to Adam and SGD optimizers in terms of rate of convergence. Due to the good results obtained in the case of shallow deep neural networks, in this chapter an hybrid optimization method is presented to train convolutional neural networks.

This method consists in combining two optimization algorithms, i.e. GNM and Adam or GNM and SGD, during the training phase within the layers of the Neural Network. In this case a convolutional neural network is considered and its parameters are updated with a different optimization algorithm.

In particular, the convolutional layers are updated with first order optimization algorithms, while the last fully connected layers are updated with the GNM method. This choice aims to combine the speed of convergence of the second order optimization method under examination, i.e. GNM, with the low computational complexity of first order method that is required to train convolutional layers.

In fact, as already described, the weight update formula of the GNM method depends on the pseudoinverse of the input matrix of the layer. It follows that in the case of convolutional neural networks, where the trainable variables are tensors, it can be computationally expensive apply the Eq. (7) to update the weights. Conversely, for the last fully connected layers of the neural network this operation can be easily computed.

To the aim to test the performances of the GNM algorithm in the hybrid optimization approach proposed, a simple CNN to classify MNIST dataset has been build with a similar approach considered in [28].

As shown in the Fig. 6, the considered CNN is composed by a convolutional layer with 32 filters of dimension 3×3 with activation function ReLU, followed by a max pooling operation with filters of 2×2 dimension. Then, the fully connected layers are applied: after the flatten layer, a dense layer of 128 units is present. Finally, the last classification layer of 10 units follows with activation function softmax.

In Fig. 7 the performances of the hybrid approach considering SGD and GNM are compared with respect the performance obtained by considering the single GD optimizer to update all the parameters of the neural network. In this case, two different learning rates have been selected for the convolutional block and the fully connected layers of the neural network.

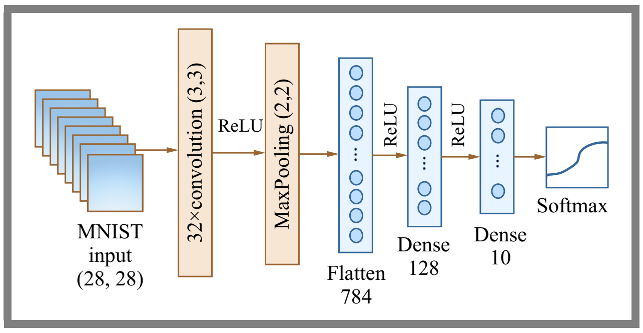

Fig. 6. MNIST dataset: convolutional neural network architecture.

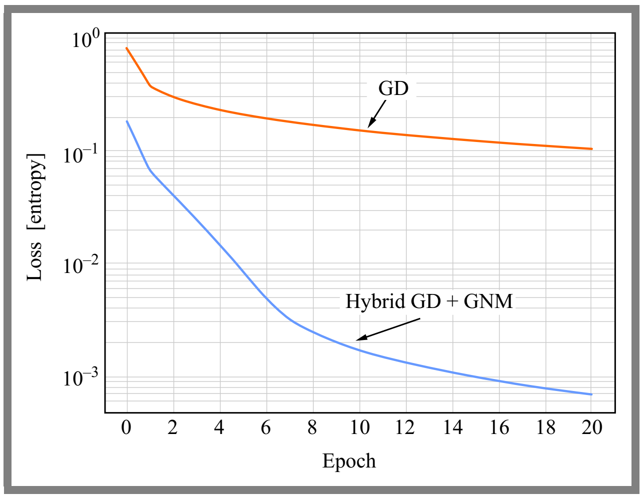

Fig. 7. MNIST dataset: comparison between fully GD and hybrid GD and GNM.

In particular, after a tuning activity aimed at obtaining the optimal η value, the learning rate for the convolutional layers trained with SGD has been fixed to 0.001, while the learning rate to update the fully connected layers and trained with GNM has been fixed to 17.

Indeed, as already highlighted, in case of GNM, the learning rate parameter depends on the condition number of the input matrix, i.e. on the order of magnitude of the inputs, see Eq. (6).

As can be seen from Fig. 7, the hybrid approach has shown the best performances in terms of rate of convergence in reaching the lower minima. This confirms the initial hypothesis, i.e. the combination of the second order method convergence rate with the scalability of first order methods would produce better performances.

In Fig. 8 the performances of the hybrid approach considering the combination of Adam and GNM optimizer are compared with respect to applying the single Adam optimizer to whole neural network. Also in this case, two different learning rates have been selected for the convolutional block and the fully connected layers of the neural network.

In particular, after a tuning activity, the learning rate for the convolutional layers trained with Adam has been fixed to 0.001, while the learning rate to update the fully connected layers and trained with GNM has been fixed to 17.

Also in this case, the hybrid approach has shown the best performances in terms of rate of convergence to reach the lower minima.

In Fig. 9 the performances of the hybrid approach considering Adam and SGD in combination with GNM in the fully connected layers are compared. One may notice, the combination of Adam and GNM optimizer shows the best performances in terms of rate of convergence in reaching the lower minima.

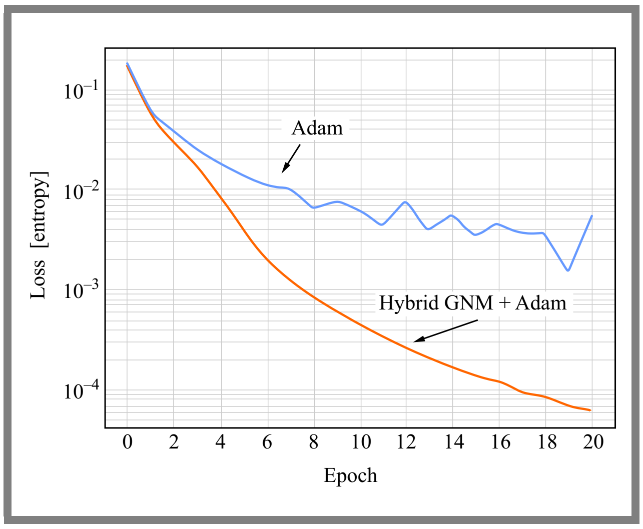

Fig. 8. MNIST dataset: comparison between fully Adam, hybrid Adam, and GNM.

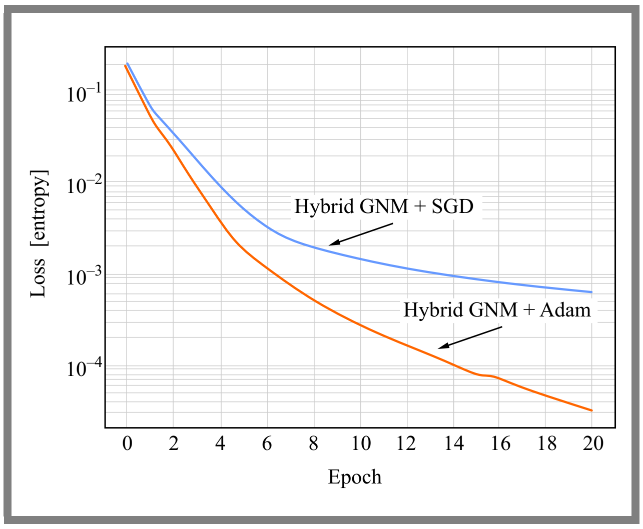

Fig. 9. MNIST dataset: comparison between hybrid approach using Adam and SGD in the convolutional layer.

5. Conclusions

In this work we presented a generalized optimization algorithm for feedforward deep neural networks and compared its performance to those of most popular algorithms in machine learning. Two popular classification tasks were considered, namely based on the MNIST and SUSY datasets. In both cases, the proposed approach showed optimal results in terms of speed of convergence, thus offering a viable alternative to most used first order methods in deep learning problems.

Given the good performances obtained with the two classification tasks considered, an hybrid optimization approach has been proposed. This new approach is configured as a powerful optimization method, as it combines the high rate of convergence of the GNM method with the scalability typical of first order ones. Also this case and as expected, the hybrid approach has shown an higher rate of convergence with respect of the standard one that considers only Adam or SGD optimizer for the training phase.

As future studies, it is essential to explore the performance of the optimizer on more diverse and challenging datasets, as well as with larger-scale deep learning architectures. This will help validate its effectiveness in real-world scenarios and provide insights into its generalizability.

Additionally, incorporating comparative studies with other state-of-the-art optimizers commonly used in deep learning can offer a more comprehensive understanding of its strengths and limitations.

In conclusion, the tested optimizer has demonstrated promising results in training deep shallow neural networks, showcasing its potential to improve convergence and achieve higher accuracy.

By addressing future challenges and exploring its application to real-world problems, the proposed method holds considerable prospects for advancing the field of deep learning and finding practical solutions in diverse domains.

References

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep Learning", *Nature*, vol. 521, pp. 436–444, 2015 (`https://doi.org/10.1038/nature14539`).

[2] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, MIT Press, 2016 (`http://www.deeplearningbook.org`).

[3] Y. Bengio, Y. LeCun, and G. Hinton, "Deep Learning for AI", *Communications of the ACM*, vol. 64, no. 7, pp. 58–65, 2021 (`https://doi.org/10.1145/3448250`).

[4] C.M. Bishop, *Neural Networks for Pattern Recognition*, Oxford University Press, 502 p., 1996 (ISBN: 9780198538646).

[5] R.D. Reed and R.J. Marks, *Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks*, MIT Press, 1999 (`https://doi.org/10.7551/mitpress/4937.001.0001`).

[6] R. Parisi, E.D. Di Claudio, G. Orlandi, and B.D. Rao, "Fast Adaptive Digital Equalization by Recurrent Neural Networks", *IEEE Transactions on Signal Processing*, vol. 45, no. 11, pp. 2731–2739, 1997 (`https://doi.org/10.1109/78.650099`).

[7] R. Battiti, "First- and Second-order Methods for Learning: Between Steepest Descent and Newton's Method", *Neural Computation*, vol. 4, no. 2, pp. 141–166, 1992 (`https://doi.org/10.1162/neco.1992.4.2.141`).

[8] L. Bottou, F.E. Curtis, and J. Nocedal, "Optimization Methods for Large-scale Machine Learning", *SIAM Review*, vol. 60, no. 2, pp. 223–311, 2018 (`https://doi.org/10.1137/16M1080173`).

[9] J. Nocedal and S.J. Wright, *Numerical Optimization*, Springer, 664 p., 2006 (`https://doi.org/10.1007/978-0-387-40065-5`).

[10] A.S. Berahas, M. Jahani, P. Richtárik, and M. Takác, "Quasi-Newton Methods for Machine Learning: Forget the Past, Just Sample", *arXiv*, 2019 (`https://doi.org/10.48550/ARXIV.1901.09997`).

[11] D. Goldfarb, Y. Ren, and A. Bahamou, "Practical Quasi-Newton Methods for Training Deep Neural Networks", *arXiv*, 2020 (`https://doi.org/10.48550/arXiv.2006.08877`).

[12] A.S. Berahas, R. Bollapragada, and J. Nocedal, "An Investigation of Newton-Sketch and Subsampled Newton Methods", *Optimization Methods and Software*, vol. 35, no. 4, pp. 661–680, 2020 (`https://doi.org/10.1080/10556788.2020.1725751`).

[13] A.S. Berahas and M. Takác, "A Robust Multi-batch L-BFGS Method for Machine Learning", *Optimization Methods and Software*, vol. 35, no. 1, pp. 191–219, 2020 (`https://doi.org/10.1080/10556788.2019.1658107`).

[14] J.E. Dennis, Jr. and J.J. Moré, "Quasi-Newton Methods, Motivation and Theory", *SIAM Review*, vol. 19, no. 1, pp. 46–89, 1977 (`https://doi.org/10.1137/1019005`).

[15] Z. Yao *et al.*, "ADAHESSIAN: An Adaptive Second Order Optimizer for Machine Learning", *arXiv*, 2020 (https://doi.org/10.48550/ARXIV.2006.00719).

[16] R. Anil *et al.*, "Scalable Second Order Optimization for Deep Learning", *arXiv*, 2020 (https://doi.org/10.48550/ARXIV.2002.09018).

[17] J.D. Lee *et al.*, "First-order Methods Almost Always Avoid Saddle Points", *arXiv*, 2017 (https://doi.org/10.48550/ARXIV.1710.07406).

[18] R. Parisi, E.D. Di Claudio, G. Orlandi, and B.D. Rao, "A Generalized Learning Paradigm Exploiting the Structure of Feedforward Neural Networks", *IEEE Transactions on Neural Networks*, vol. 7, no. 6, pp. 1450–1460, 1996 (https://doi.org/10.1109/72.548172).

[19] S. Ruder, "An Overview of Gradient Descent Optimization Algorithms", *arXiv*, 2016 (https://doi.org/10.48550/ARXIV.1609.04747).

[20] H. Robbins and S. Monro, "A Stochastic Approximation Method", *The Annals of Mathematical Statistics*, vol. 22, no. 3, pp. 400–407, 1951 (https://doi.org/10.1214/aoms/1177729586).

[21] R. Rojas, *Neural Networks. A Systematic Introduction*, Springer, 504 p., 2006 (https://doi.org/10.1007/978-3-642-61068-4).

[22] D.E. Rumelhart and J.L. McClelland, "Learning Internal Representations by Error Propagation", in: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations*, MIT Press, pp. 318–362, 1987 (ISBN: 9780262291408).

[23] D.P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization", *arXiv*, 2014 (https://doi.org/10.48550/ARXIV.1412.6980).

[24] J.D. Lee *et al.*, "Basic Classification: Classify Images of Clothing" (https://www.tensorflow.org/tutorials/keras/classification).

[25] Y. LeCun, C. Cortes, and C.J.C. Burges, *The MNIST Database of Handwritten Digits*, 2012 (http://yann.lecun.com/exdb/mnist/).

[26] M. Abadi *et al.*, "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems", *arXiv*, 2016 (https://doi.org/10.48550/ARXIV.1603.04467).

[27] P. Baldi, P. Sadowski, and D. Whiteson, "Searching for Exotic Particles in High-energy Physics with Deep Learning", *Nature Communications*, vol. 5, art. no. 4308, 2014 (https://doi.org/10.1038/ncomms5308).

[28] D.-Y. Ge *et al.*, "Design of High Accuracy Detector for MNIST Handwritten Digit Recognition Based on Convolutional Neural Network", *2019 12th International Conference on Intelligent Computation Technology and Automation (ICICTA)*, Xiangtan, China, 2019 (https://doi.org/10.1109/ICICTA49267.2019.00145).

Francesca Ponti, Ph.D.
Department of Information Engineering, Electronics and Telecommunications
https://orcid.org/0000-0003-1855-7280
E-mail: francesca.ponti@uniroma.it
"Sapienza" University of Rome, Rome, Italy
https://www.uniroma1.it/en/

Fabrizio Frezza, Prof.
Department of Information Engineering, Electronics and Telecommunications
https://orcid.org/0000-0001-9457-7617
E-mail: fabrizio.frezza@uniroma1.it
"Sapienza" University of Rome, Rome, Italy
https://www.uniroma1.it/en/

Patrizio Simeoni, Ph.D.
Department of Electronic and Communications Engineering
https://orcid.org/0009-0008-7365-7624
E-mail: patrizio.simeoni@setu.ie
South East Technological University, Carlow, Ireland
https://www.setu.ie

Raffaele Parisi, Prof.
Department of Information Engineering, Electronics and Telecommunications
https://orcid.org/0000-0002-6062-0274
E-mail: raffaele.parisi@uniroma1.it
"Sapienza" University of Rome, Rome, Italy
https://www.uniroma1.it/en/