Forward-Forward with Low Memory Matrix Adaptation Exploiting Gradient Information

Riccardo Lunelli, *Dipartimento di Ingengeria e Scienza dell' Informazione, Università degli studi di Trento*

***Abstract*—The aim of this project is to explore the application of the Low-Memory Matrix Adaptation Evolution Strategy[1] in training a neural network with the new Forward-Forward paradigm proposed by Hinton in late 2022[2]. The discussion begins with an introduction to the problem and how LMMA-ES is integrated with the Forward-Forward paradigm. A proposal to include gradient information within the LMMA-ES algorithm is then introduced, along with ways to exploit this information. Experimental results highlighting both the advantages and limitations of this approach are presented in the subsequent sections. The work concludes with a discussion of the implications, potential applications, and future enhancements of this method.**

I. Introduction

THE Forward-Forward algorithm[2], recently introduced as an alternative to the widely used Back Propagation, proposes a novel approach for neural network training. The concept behind this approach is to simulate the learning process of the brain more closely than what happens with the Back Propagation where the learning procedure is composed of a Forward-Backward mechanism. Here instead the information is passed only forward, so the data is processed one layer at a time. This paradigm necessitates the creation of both positive and negative samples, with the labels embedded within the data itself using one-hot encoding. As suggested by Hinton in the original paper, the first ten pixels of the image from the MNIST dataset are replaced with the one-hot encoding label, and for negative data, a random one-hot encoding label is used. The paradigm's ability to train each layer independently reduces computational demands compared to traditional Back Propagation methods.

Training neural networks with Genetic Algorithms poses significant challenges, particularly for larger networks. We can consider finding the best weights for a given architecture as a Large-Scale Problem. This work leverages the Forward-Forward paradigm and the principle of "divide and conquer" to simplify the problem, employing Genetic Algorithms to train each layer individually.

The Low-Memory Matrix Adaptation (LMMA)[1], a variant of the Covariance Matrix Adaptation, has been chosen for training due to its reduced complexity and state-of-the-art performance in Evolution Strategies. This variant reduces the complexity of the Covariant Matrix Adaptation using an estimation of the covariance matrix and employs the Cumulative Step-size Adaptation [3]. The Covariance Matrix Adaptation is an Evolution Strategy where offsprings are generated randomly using a covariance matrix that is learnt during the generation update. The rule of creation of an individual x_i is the following:

$$x_i = y + \sigma * \mathcal{N}(0, C)$$

Where y is a weighted mean of μ best individual in the previous step, σ is the step size and C is the covariance matrix. The LMMA-ES has only a different way to estimate C but the principle is the same. The Cumulative Step-size Adaptation allows the algorithm to learn the best step size σ increasing it when recent steps are in the same direction and decreasing it when steps tends to have different directions.

Two variants of the LMMA-ES will be introduced, one that incorporates gradient information and another that uses an estimation of the gradient. This study utilizes the MNIST and CIFAR10 datasets to validate the proposed algorithms, with the `deap` and `torch` Python libraries serving as the foundational tools.

II. Forward-Forward

The initial step of this project involved establishing a baseline for algorithm testing. The starting point was a publicly available repository found on GitHub[4]. Necessary modifications were made to this codebase to facilitate the injection of the algorithm for individual layer training. Subsequently, the loss and the method for label information injection, as proposed in the original paper, were replaced with the approaches suggested in the SymBa algorithm[5] to expedite convergence. Accordingly, the loss function used was:

$$L_{SymBa} = log(1 + e^{-\alpha\Delta})$$

where $\alpha = 4$ and :

$$\Delta = G_{pos} - G_{neg}$$

$$G_{pos} = \sum_j y_{pos,j}^2$$

$$G_{neg} = \sum_j y_{neg,j}^2$$

where y_{pos} and y_{neg} are the activations of the neurons within the positive and the negative samples respectively.

Rather than substituting the one hot encoding of the class inside the image itself, the information was concatenated. This modification was required due to the SymBa proposal potentially increasing the search space excessively. To be precise, to 28*28 (784) in the case of MNIST and 32*32 (1024) for CIFAR10.

III. LMMA-ES

A. Vanilla version

The second step involved implementing the algorithm as proposed in the original paper[1]. This was accomplished using PyTorch, which allowed for more efficient utilization of the GPU or Neural Engine during the execution of the algorithm. Results are detailed in subsequent sections. A Hall of Fame mechanism was employed in this and all other variants to keep track of the best solution found.

B. Gradient Exploitation

The authors of the LMMA-ES paper suggest using momentum information based on the gradient to enhance results. This information was incorporated by calculating the gradient for the variable y at each update step. The gradient was then transformed into a momentum vector using the AMSGrad variant of the Adam algorithm, which offers improved estimation of the direction. Following this, noise was added to the momentum vector and normalized. Normalization was necessary to maintain the resulting vector's standard deviation at 1.

$$z_i = momentum + noise_i$$

$$z_i = z_i / z_i.std()$$

It should be noted that z_i does not have zero mean, as it is biased in the direction of the momentum. The goal of this method is to bias the creation of new individuals towards a more optimal point in space. Thus, the formula for the creation of new individuals becomes:

$$x_i = y + \sigma * \mathcal{N}(momentum, C)$$

C. Gradient Estimation

This method generalizes the algorithm for problems where gradient information is not readily available. When direct gradient calculation is unfeasible, strategies for gradient estimation can be used. Although an estimated gradient might lack the precision of a directly calculated one, it can still improve upon the vanilla LMMA-ES. The chosen strategy for gradient estimation leverages information provided by the individuals, necessitating only the calculation of the fitness value for y. Only the top μ individuals are considered for the gradient estimation. The first step involves calculating the direction from y to the individual:

$$dir_i = y - ind_i$$

Considering only the best μ individuals is beneficial as it provides more insight into the optimal direction, especially in cases involving a high number of variables. Next, the difference in fitness values between y and the i_{th} individual is calculated:

$$f_i = fit_y - fit_{ind}$$

The direction is then multiplied by the difference in fitness values. To penalize distant individuals that could result in poor gradient estimation, the result is divided by the squared norm of the direction:

$$grad_i = (dir_i * f_i) / ||dir_i||^2$$

The final step is to calculate the mean of all estimated gradients:

$$gradient = 1/\mu * \sum_{i}^{\mu} grad_i$$

Considering the possible inaccuracy of estimated gradients, it's crucial to prevent biasing in all individuals to a poor estimation.

D. Lévy Flight

The previously described method sometimes exhibits stability issues due to the biasing by an inaccurate estimation of the gradient. To increase randomness on the gradient and promote exploration, a Lévy Flight mechanism was incorporated[6]. This mechanism allows random weights to be assigned to the momentum vector, with the weights being drawn from a Lévy distribution. Consequently, the weights are mostly situated between 0 and 1, but occasionally they can attain very high values.

Conceptually, this permits the exploitation of gradient information when it offers a reliable estimation, and populations with higher weights will be favored. Conversely, if the estimation is poor, the algorithm will select individuals with lower weights for the gradient, effectively reverting to the vanilla implementation.

The Lévy Flight is applied prior to the addition of noise to the gradient:

$$momentum_i = momentum * |levy\_flight()|$$

Subsequently, z_i is calculated using the i_{th} momentum:

$$z_i = momentum_i + noise_i$$

Finally, it is normalized as per the previous section:

$$z_i = z_i * z_i * std()$$

IV. Results

A. MNIST Dataset

This algorithm was evaluated using the MNIST Dataset with a simple two-layer network configuration: the first layer has dimensions 794x512, and the second layer 512x316. The value 794 is determined by the size of the image (28*28) plus the one-hot encoding for the class. I used a momentum value of 0.6 because the network was not trained using batches, thus, I expected the gradient not to be excessively noisy.

As depicted in Fig.1, the most effective algorithm is the one exploiting the actual gradient. There is no significant performance difference observed when using Lévy Flight. The versions using estimated gradient demonstrate faster convergence than the Vanilla version, and the implementation of Lévy Flight enhances the convergence speed. This finding is of significant interest because it enables the generalization of this algorithm to other scenarios where the gradient calculation is not possible.

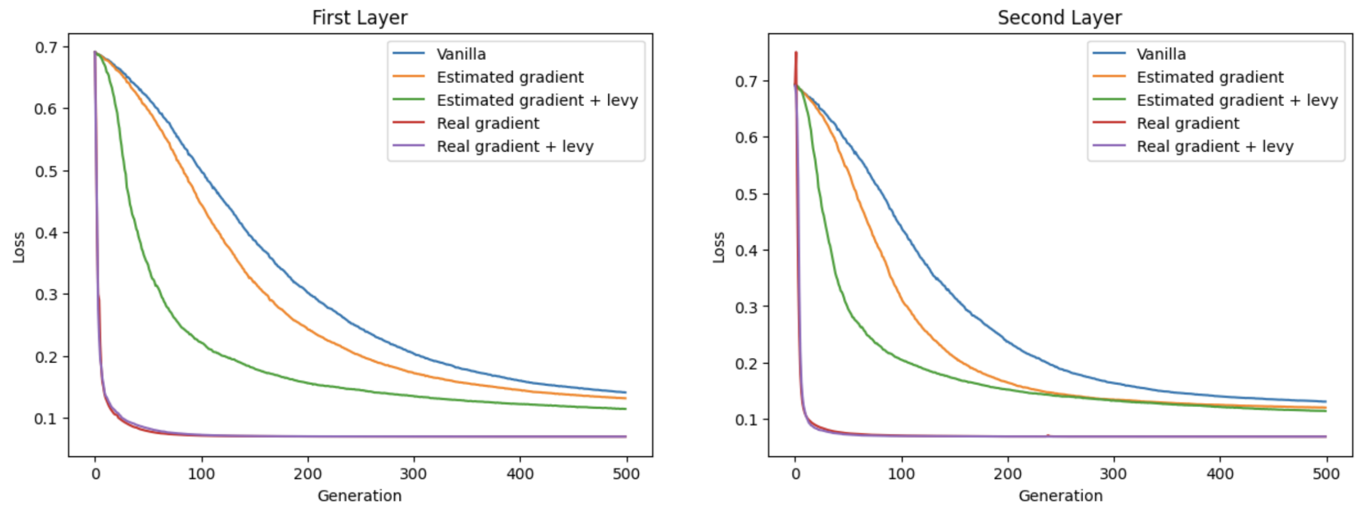

Fig. 1. Results on the MNIST dataset for the Vanilla LMMA-ES and the proposed variants for the first and the second layer.

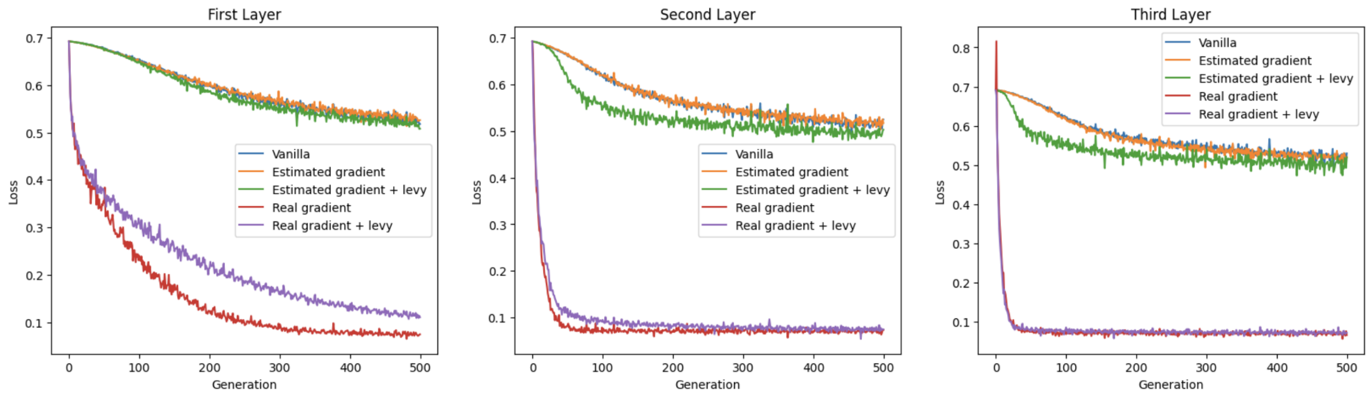

Fig. 2. Results on the CIFAR10 Dataset for the Vanilla LMMA-ES and the proposed variants for the first and the second layer. The Vanilla is almost covered by the version with Estimated Gradient

TABLE I
ACCURACY ON TEST DATASET

	Vanilla	Estimated Gradient	Estimated Gradient + Lévy	Real Gradient	Real Gradient + Lévy	Forward Forward
MNIST	87.9%	89.4%	86.7%	**95.7%**	95%	94.1%
CIFAR10	31.7%	32.0%	34.0%	**47.2%**	47.0%	41%

B. *CIFAR10 Dataset*

This algorithm was tested on the CIFAR10 Dataset using a simple three-layer network configuration: the first layer of 3082x1024, the second of 1024x500, and the third of 500x500. The first layer takes into account the three channels plus the one-hot encoding vector. In this scenario, the network was trained with batches of data of size 4096, so a default momentum value of 0.9 is used in order to have a more stable gradient.

As shown in Fig.2, the results align closely with those obtained using the MNIST Dataset. The variants employing the actual gradient clearly outperform the others. Here, it becomes more evident that Lévy Flight does not enhance the quality of good gradient information, but it does improve the estimated version. This finding further substantiates that Lévy Flight can enhance the estimated gradient when its accuracy is lacking. An interesting discrepancy with the MNIST dataset is that the version utilizing only estimated gradient and the vanilla version seem to have similar behavior. This might be due to the fact that the estimated gradient is very small, rendering the contribution to the creation of our z_i nearly negligible.

C. *Accuracy*

Table I presents the overall accuracy of the two networks on the test datasets. As seen so far, the method utilizing the real gradient consistently delivers the highest accuracy. Interestingly, the Vanilla version shows better accuracy than the Estimated Gradient version with Lévy Flight for the MNIST dataset. However, these observations, and any others we could potentially make by examining the accuracy, do not significantly impact the objectives of this work. They could be attributed to issues related to overfitting or randomness, which have not been addressed in this study.

With regard to the original Forward Forward algorithm, it ran for 500 epochs, consistent with all other methods. It's interesting to note that the versions exploiting the real gradient outperform the legacy Forward Forward method. However, as previously mentioned, there could be other underlying issues affecting these results, and more investigations would be necessary to establish the statistical significance of these findings.

D. Cost

Each variant of the algorithm underwent training for 500 generations per layer, with every generation evaluating a population ranging from 40 to 50 individuals (larger layers corresponded to larger population sizes). The legacy Forward-Forward was also trained over 500 epochs. However, for the CIFAR10 dataset, no batches were used in this case. Consequently, the number of evaluations is significantly higher compared to the legacy Forward-Forward method, leading to a substantially longer execution time.

V. Conclusion

The results obtained in this work offer intriguing insights and provide a novel approach to training neural networks using Genetic Algorithms. More importantly, they explore how to introduce gradient information into LMMA-ES. The primary takeaway from this study is the revelation that LMMA-ES can be significantly enhanced with gradient information. Most crucially, gradient information can be approximated when it's not directly available.

In this work, I have used one method for calculating the gradient, but alternative estimators can be employed to enhance the performance of the algorithm. As demonstrated with the real gradient estimation, better estimations can lead to faster convergence.

Future research should aim to benchmark this algorithm on other problems where the direct calculation of the gradient is not possible, and subsequently compare the performance with the Vanilla version. Moreover, investigating ways to improve or consolidate this method could be beneficial, as the use of other estimation techniques could potentially enhance performance even further.

The primary challenge with this new algorithm, however, is the lack of theoretical support, particularly concerning the biasing of of z_i by the gradient information. This indicates a need for additional investigations to bolster understanding and refine this approach. Understanding the fundamental aspects of the method better could also provide insights into ways of extending the approach or potentially applying it to other related problems. This would further contribute to the body of knowledge on this topic, providing a stronger foundation for the application and further development of the algorithm.

References

[1] I. Loshchilov, T. Glasmachers, and H.-G. Beyer, "Large scale black-box optimization by limited-memory matrix adaptation," *IEEE Transactions on Evolutionary Computation*, vol. 23, no. 2, pp. 353–358, 2019.

[2] G. Hinton, "The forward-forward algorithm: Some preliminary investigations," 2022.

[3] A. Ostermeier, A. Gawelczyk, and N. Hansen, "Step-size adaption based on non-local use of selection information." vol. 866, 10 1994, pp. 189–198.

[4] J. Y. Mohammad Pezeshki, Haaris Rahman, "pytorch_forward_forward," https://github.com/mohammadpz/pytorch_forward_forward.

[5] H.-C. Lee and J. Song, "Symba: Symmetric backpropagation-free contrastive learning with forward-forward algorithm for optimizing convergence," 2023.

[6] Wikipedia, "Lévy flight, wikipedia, the free encyclopedia." [Online]. Available: https://en.wikipedia.org/wiki/Lévy_flight