Exploration of the evolutionary mutation footprint on Generative Adversarial Networks

Pawel Kopciewicz
AGH University of Science and Technology
Krakow, Poland, al. Mickiewicza 30
Email: Pawel.Kopciewicz@fis.agh.edu.pl

Vitalii Morskyi
Rzeszow University of Technology
Rzeszow, Poland, al. Powstancow Warszawy 12
Email: Vitalij20358@gmail.com

***Abstract*—In the last few years, the Generative Adversarial Networks (GAN) proved to be an excellent tool for image generation and have found a variety of practical applications. In this paper, the specific evolutional footprint for GAN generator training improvement is proposed and examined. The idea behind the research is that the generator's weights learning at the constant discriminator can be optionally tackled as a complex optimization problem. Although the fitness function for the generator cannot be straightly constructed, its local state can be evaluated in the function of the discriminator performance. One can assume that the fitness function may have an unknown but potentially high number of local minima, which can restrict the gradient descent to move towards some directions. The vast number of degrees of freedom usually seen in the image processing layers exceeds the standard problems for intelligent optimization algorithms, therefore the hybrid combination of the gradient descent with the selected parts of differential evolutionary algorithms, one of the best optimization methods in the field, are considered.**

I. Introduction

The Generative Adversarial Networks (GANs) [1] are recently one of the most investigated machine learning domains, with remarkable applications varying from computer vision, semantic segmentation, and natural language processing. Their recent state-of-art review with lately advances on the generative models can be found in [2] [3].

This paper investigates the possible applications of various evolutional approaches and their footprint on the generative network training, especially the mutation phenomena. The mutation is a part of the Genetic Algorithm (GA) family, which originated long before the GANs [5] and become very popular and researched area [6] forming the ground for other optimization methods based on nature inspiration [16], strongly developed in the recent years [17]. In particular, the Differential Evolutionary (DE) algorithms [18], which originated from GA [19] and continuously developed [20] in various directions [21] [22] [23], proved to be a powerful tool for various optimization problems [24] at both single and multi-objective optimization [25]. The idea of acquiring the evolutional training was present in the neural networks before creation of GAN [7] [8] with alternative training by GA [9]. Since that time, various modifications [10] and applications can be spotted, e.g. time series forecasting [11] [12]. Even though the GA alone have much less training capabilities [13], the hybrid models can outperform the usual back-propagation approach [14] [15]. The adoption of GA and later DE was wide and reached the point that the whole structure of the generator in a generative model obeyed to a GA optimization, dubbed as Evolutional GAN [26]. The multiple improvements were proposed [27] [28], including the multi-objective optimization [29] or even using GANs for optimization other problems [30].

The paper focuses on various approaches of mutations and their footprint on the generetor training. The hybrid combined models of DE in the GAN training in this paper is to use the mutation and selection step on the generator weights to support it at improving its loss function. The mutation role in optimization algorithm in general is to cope with local minima attractivenes, whilst the actual optimization role is carried out primarily by selection [31]. There are many mutation strategies that can be found in literature, e.g. mutation based on displacing, inversing, scrambling, bit flipping or reversing [32]. Their actual usage and performance were widely tested [32] [33]. Various mutation strategies were applied to specific differential evolutionary models [34], i.e. for solving global optimization problems [35], which even emphasized their role in the optimization process to be an actual impactful on the optimization [36] [37]. The mutation approach investigated in this paper is a bitwise mutation for the floating number with adjustable properties over time and symmetric masks applied to the mutated weights. The paper draws out the possible yield on the generator training to support the gradient algorithm and investigates the actual mutation footprint with respect to the discriminator. It makes an argument that a good suited mutation strategy can support the GAN training more than it could result from the random improvement caused by random processes. The image material for GAN is a mnist digits databank. The improvement on the image generation is particularly visible on the best generated digits, that have the significantly higher quality to the best digits generated without mutations, however, with the bad images quality and their ratio not changed for both models, that would not happen if mutation procedure was granting a random impact on the generative model.

The paper is constructed as follows. The evolutional concepts are thoroughly described in Section II, along with the expected impact on the generative model. The results are discussed in Section IV. They are preceded by the brief description of the computing and experimental setup in Sec-

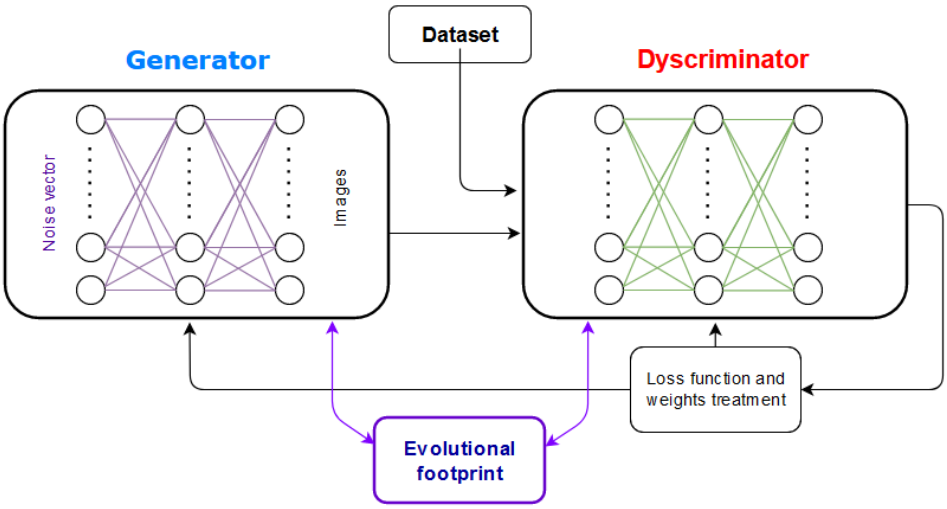

Fig. 1. The GAN structure with used in the studies.

tion III. The paper is finished by the conclusions and proposed applications in Section V.

II. Evolutional training concepts

The mutation strategy was adjusted for the CPU maximal efficiency to mitigate high time consumption processes in between the GAN training algorithm.

The type of the weights is a numpy float64, a type of 64 bit precision. The general representation within the given endianness of a binary word includes the first bit standing for signedness, the next evelen for exponent's power and the rest for the mantissa factor [38]. The vast number of weights (mutation candidates) can be costly in terms of CPU calculations and an efficient, specific masking strategy for mutation has to be arranged. When the bits is masked, no mutation is taking place. Usually the mutation likelihood of each bit ranges from 0.01% to 5% what is also described later in this Section, so the vast majority of the bits should always be masked. Additionally, the exponent's bit word is always masked unless the optional exponent mutation is allowed. However, given the nature of the non-linear exponent factor, no particular advantage is achieved, while the memory cost of the mutation process would be higher by 33%. The signedness bit is always masked. For the maximal CPU efficiency, it was decided to use following masking strategy. The masks are stored in three 16-bit types (unsigned integers) after each bit's value is rolled. This gives the coverage of 48 bits of the mantissa and remaining 3 least significant bits never classified for the mutations.

In generating mutations, two main approaches can be used to create new populations: binary and floating-point coding. Nowadays, the latter one is more preferable, but in our research we have chosen the binary way. To begin with, we generate the mutation mask, where 0 means *"leave the bit unchanged"* and 1 – *"swap the bit"*. Creating those masks bit by bit can become very time consuming and create a bottleneck of our model, so we use another algorithm. Assuming that p is the probability of the single bit to be changed, we calculate the probabilities for the chunks with a certain number of bits n_b. For example if $p = 0.2$ and $n_b = 4$ the chance that only second bit in that chunk changes is $0.2^1 \cdot 0.8^3 = 10.24\%$. We can match this probability to the following binary number $4_{(10)} = 0100_{(2)}$, which represents a part of our mutation mask. Following that, we can calculate such a chance for each possible chunk and randomly choose chunks instead of single bits. To create the mutation mask, we only have to concatenate those chunks by multiplying each chunk by 2^k and adding them all up. Here k stands for the number of bits the particular chunk has to be shifted to the left. For example, if we have sampled two chunks $4_{(10)} = 0100_{(2)}$ and $9_{(10)} = 1001_{(2)}$ in order to generate 8 bit mask like this one: 0100 1001, we multiply the first chunk by $2^4 = 16$ and add the result to the second chunk: $16_{(10)} \cdot 4_{(10)} + 9_{(10)} = 73_{(10)} = 0100\ 1001_{(2)}$.

Finally, we apply the resulting mutation mask over the original number represented in the IEEE 754 double-precision binary floating-point format using exclusive OR logical operator (XOR):

$$mutation_i = mask_i \oplus original\ value$$

The aforementioned approach improves time complexity by 150 times on arrays with 500K numbers. To maximize the use of commonly available AVX extensions in modern CPUs, everything is implemented in the vectorized form giving another ten times improvement on the same dataset, making the algorithm 1500 times faster than its basic version.

At the studies the mutation setup was tested with various criteria, e.g. the optimal mutation frequency, which is the number of training epochs after mutation can happen. Optionally after each epoch a mutation is directed with a certain probability. When the mutation is announced, the weights are extracted from the GAN into a CPU part of code that further executes the mutation algorithm. For each float in the weight vectors,

a probability check is followed by the qualification into a mutation. When positive, three 16-bit mask words are built. The factor classifying the weight to mutate was checked in the range of 5 - 20%, while the mutation likelihood for a single bit was 0.001 - 5%. It was also studied whether the floating likelihood, such like the increasing ratio of overall mutated weights affect the results, what is shown and described in Section IV.

While the cross-over was neglected as its role overlaps with the gradient descent already existing in the GAN training, the selection process is more troublesome to tackle as the population is always obtained from one particular weights set after a bitwise mutation. Although the multiplication factor ϵ mostly leads to the unique solution, at least at the low significant bits range, the selection process assumes that the parents to the solutions also come from the older selection, which is not true in this specific case. One can consider the single selection step an initial arrangement of the vectors, which would rather mimic the other mutation approach instead than play its optimization role. Therefore a multiple step selection was implemented with the generator fitness check each the new population is generated. The weights obtained in the youngest generation have the priority to replace the GAN weights, in case the population have no suitable solutions, the parent population is checked instead. When neither the final population nor the parent population were able to yield the candidate, the GAN is instructed to train its usual way ahead. The impact of the evolutional behavior of generator's weights has to be kept relatively low to avoid the generator outperforming the discriminator, disturbing the training process. The methods described above have a medium impact on the computational time of algorithm's execution. The calculation cost of the three-fold differential optimization will eventually depend on the mutation frequency and the number of weights in vectors to optimize. For instance, 1.5 million weights and the mutation procedure after every 200 batches of 32 samples consumes around 50% more memory.

The weights quality is evaluated using the current discriminator state in similar way to the general generator feedback when its loss function is calculated. As already mentioned, the fitness function cannot be straightly constructed because of the subjective generator output, the condition of the generator can be measured indirectly. Plenty of methods are available, in this particular research the discriminator check was used. Provided the discriminator trains with the relative same pace to the generator, it can tell the prediction that should be less accurate for the slightly improved generator. Since the mutation frequency is usually not higher than one after 200 batches, it does not deregulate the generator and discriminator performance. The expected result of the studies is to prevent the generator from stacking in the local minima, that would manifest in approaching the satisfactionary results faster than before. The entire code of the project is available on github in one of the authors' repository (cite).

III. The network setup

IV. Results

The variants described in previous Section were tested and their impact on the overall GAN performance investigated. The first entity to study was the generic impact of the mutation probability p to the generator behavior, including the zero probability $p = 0$. The mutation was repeated 50 times to generate 50 different sets of weights. For each weight set the generator performance was evaluated using the current state of the discriminator.

Regardless the mutation likelihood, the single mutated set before selection was usually able to generate a better solution, which statistically reduced the gan loss function by 10% on average. It was improved to around 20% on average with the three-fold selection on top of the mutation. The first tested probability of the mutation likelihood $p = 0.05$ was providing better results for 20 epochs (each epoch of 1024 batches of 32 images). This value of p was very high, as for 48 bits in each weight the overall probability that at least mutation will take place is 91.4% for each weight. The vector of weights is mostly changed and the solution can be far away from the actual initial position. During first 20 epochs GAN was trained enough that none from 50 highly mutated vectors was able to pass. As over the training, the generator weights are moving closer towards their final positions, the floating p value was proposed. The chance of each bit to mutate was inversely proportional to the epoch number. The formula was adjusted to be $p = 0.05$ for the initial value and move towards $p = 0.00001$ value when GAN reaches the farthest epochs. In this approach the mutation was almost always able to deliver the better weights set to a generator, even at the final training stages. The mutation itself leads to only minor improvements in the results. The generated digits are moving just a little bit faster towards the final state, what can be caused by the fact that the mutation process actually evaluates the discriminator loss on the generated data and thus it can count as an optimization step. With the selection added,

V. Conclusion

The evolutionary in terms of the mutation and a selection was studied for various approaches of a bitwise mutation, with various conditions investigated and the impact on the results and the overall GAN performance was measured. The whole process was optimized in terms of GPU and CPU efficiency, for models on PyTorch and Keras. In particular, the CPU calculations was modified by around 10 times to the standard mutation approach. The impact on the results was also checked. It was found that a mutation operation in a specific setup can improve the GAN results in terms of faster training and generating a higher amount of digits in better quality. However, the new version of GAN was not better in terms of generating the bad quaity results. This might be a footprint of mutation to have the GAN better performing for specific set of noise. Concluding, the GAN behaved better for image improvement but did not improved the bad generated

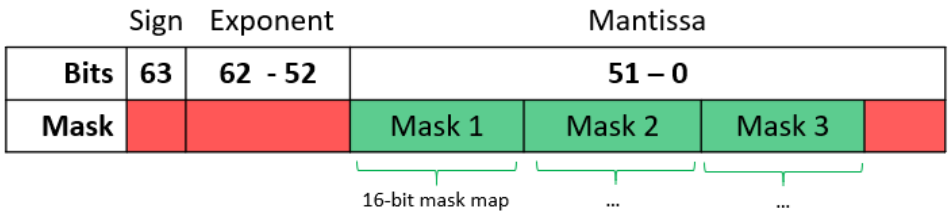

Fig. 2. Mutation strategy for CPU efficiency.

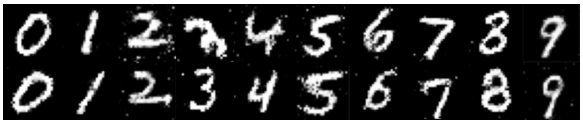

Fig. 3. Digits comparison for the original GAN (on top) and the GAN with mutations diminishing over time (on bottom) in Keras.

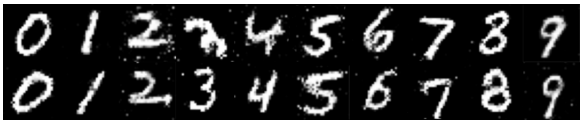

Fig. 4. Digits comparison for the original GAN (on top) and the GAN with mutations diminishing over time (on bottom) in PyTorch.

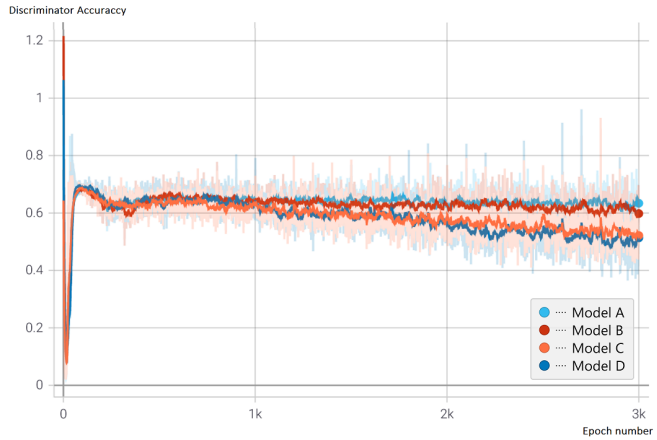

Fig. 5. Discriminator accuracy for four different bitwise mutation likelihood, A: p = 0 (no mutations); B: p = 0.0001; C: p = 0.0005; D: p = 0.001.

figures. Perhaps the idea behind the studies is more valuable and one should investigate various hybrid improvements for the generator training improving the badly generated samples with no significant impact on the CPU/GPU computational cost.

Acknowledgment

We acknowledge support from the Polish National Science Centre NCN (UMO-2020/37/N/ST2/04008).

References

[1] I. Goodfellow et al. , *Generative adversarial nets*, Advances in Neural Information Processing Systems, 2014, pp. 2672 - 2680

[2] Z. Wang et al. , *Generative Adversarial Networks in Computer Vision: A Survey and Taxonomy*, ACM Computing Surveys, 2021, Vol. 54 Is. 2

[3] Z. Pan et al. , *Recent Progress on Generative Adversarial Networks (GANs): A Survey*, IEEE Access, 2019, Vol. 7

[4] A. Aggarwal , *Generative adversarial network: An overview of theory and applications*, International Journal of Information Management, 2021, Vol. 1

[5] J. H. Holland , *Adaptation in Natural and Artificial Systems*, The University of Michigan Press, 1795

[6] A. Lambora, K. Gupta, K. Chopra , *Genetic Algorithm - A Literature Review*, 2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon), 2019, pp. 380-384

[7] D. J. Montana, L. Davis , *Training feedforward neural networks using genetic algorithms*, IJCAI'89: Proceedings of the 11th international joint conference on Artificial Intelligence, 1989, Vol. 1, pp. 762-767

[8] R. S. Sexton, R. E. Dorsey, J. D. Johnson , *Toward global optimization of neural networks: A comparison of the genetic algorithm and backpropagation*, Decision Support Systems, 1998, Vol. 22, Is. 2, pp. 171-185

[9] J. N. D. Gupta, R. S. Sexton , *Comparing backpropagation with a generic algorithm for neural network training*, Omega, 1999, Vol. 27, Is. 6, pp. 679-684

[10] H. Huang, J. Li, C. Xiao , *A proposed iteration optimization approach integrating backpropagation neural network with genetic algorithm*, Expert Systems with Applications, 2015, Vol. 42, Is. 1, pp. 146-155

[11] Haviluddin, R. Alfred , *A genetic-based backpropagation neural network for forecasting in time-series data*, 2015 International Conference on Science in Information Technology, 2015, pp. 158-163

[12] J. Gill, B. Singh, S. Singh , *Training back propagation neural networks with genetic algorithm for weather forecasting*, IEEE 8th International Symposium on Intelligent Systems and Informatics, 2010

[13] C. Chandre Gowda, S. G. Mayya , *Comparison of Back Propagation Neural Network and Genetic Algorithm Neural Network for Stream Flow Prediction*, Journal of Computational Environmental Sciences, 2014

[14] F. Yin, H. Mao, L. Hua , *A hybrid of back propagation neural network and genetic algorithm for optimization of injection molding process parameters*, Materials Design, 2011, Vol. 32, Is. 6, pp. 3457-3464

[15] M. Hu , *Optimizing Back Propagation Neural Network with Genetic Algorithm for Man-hour Prediction in Chemical Equipment Design*, Chemical Engineering Transactions, 2018, Vol. 66, pp. 877-882

[16] X. Yang , *Nature-Inspired Optimization Algorithms*, 2014 (1st edition), 2020 (2nd edition)

[17] X. Yang , *Nature-Inspired Algorithms and Applied Optimization*, 2018

[18] R. Storn, K. Price , *Differential Evolution - A Simple and Efficient Heuristic for global Optimization over Continuous Spaces*, Journal of Global Optimization, 1997, Vol. 11, pp. 341 - 359

[19] R. Storn , *On the usage of differential evolution for function optimization*, Proceedings of North American Fuzzy Information Processing, 1996

[20] Bilal et al. , *Differential Evolution: A review of more than two decades of research*, Engineering Applications of Artificial Intelligence, 2020, Vol. 90

[21] S. Hassan. et al. , *Multi-variant differential evolution algorithm for feature selection*, Scientific Reports, 2020, Vol. 90

[22] M. A. Bakar et al. , *A Customized Differential Evolutionary Algorithm for Bounded Constrained Optimization Problems*, Complexity, 2021, Vol. 2021

[23] X. Zhong, P. Cheng , *An Improved Differential Evolution Algorithm Based on Dual-Strategy*, Mathematical Problems in Engineering, 2020, Vol. 2020

[24] V. Kachitvichyanukul, *Comparison of Three Evolutionary Algorithms: GA, PSO, and DE*, Industrial Engineering and Management System, 2012, Vol. 11, pp. 215-223
[25] U. Mlakar et al. , *Multi-Objective Differential Evolution for feature selection in Facial Expression Recognition systems*, Expert Systems with Applications, 2017, Vol. 89, pp. 129-137
[26] C. Wang et al. , *Evolutionary Generative Adversarial Networks*, IEEE Transactions on Evolutionary Computation, 2019, Vol. 23, pp. 921-934
[27] J. Toutouh et al. , *Spatial evolutionary generative adversarial networks*, Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion, 2020, pp. 1824-1831
[28] V. F. Costa et al. , *Neuroevolution of Generative Adversarial Networks*, Deep Neural Evolution, 2020, pp. 293-322
[29] M. Baioletti et al. , *Multi-objective evolutionary GAN*, Deep Neural Evolution, 2020, pp. 293-322
[30] C. He et al. , *Evolutionary Multiobjective Optimization Driven by Generative Adversarial Networks (GANs)*, IEEE Transactions on Cybernetics, 2021, Vol. 51, no. 6, pp. 3129-3142
[31] S. Katoch, S. S. Chauhan, V. Kumar , *A review on genetic algorithm: past, present and future*, Multimedia Tools and Applications, 2020, Vol. 80, pp. 8091-8126
[32] K. Opara, J. Arabas , *Comparison of mutation strategies in Differential Evolution - A probabilistic perspective*, Swarm and Evolutionary Computation, 2018, pp. 53-69
[33] R. Mallipeddi et al. , *Differential evolution algorithm with ensemble of parameters and mutation strategies*, Applied Soft Computing, 2011, Vol. 11, Is. 2, pp. 1679-1696
[34] X. Zhong, M. Duan, P. Cheng , *Ranking-based hierarchical random mutation in differential evolution*, PLOS ONE, 2021, Vol. 16
[35] M. Leon, N. Xiong , *Investigation of Mutation Strategies in Differential Evolution for Solving Global Optimization Problems*, Artificial Intelligence and Soft Computing, 2014, pp. 372-383
[36] T. Wang et al. , *Adaptive Dynamic Disturbance Strategy for Differential Evolution Algorithm*, MDPI Applied Science, 2020
[37] C. Lu, S. Chiu, C. Hsu, S. Yen , *Enhanced Differential Evolution Based on Adaptive Mutation and Wrapper Local Search Strategies for Global Optimization Problems*, Journal of Applied Research and Technology JART, 2014, Vol. 12, Is. 6, pp. 1131-1143
[38] *IEEE Standard for Floating-Point Arithmetic*, IEEE Std 754-2008, 2008, Vol. 12, Is. 6, pp. 1-70