2021 20th IEEE International Conference on Machine Learning and Applications (ICMLA) | 978-1-6654-4337-1/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICMLA52953.2021.00158

Pruned Genetic-NAS on GPU Accelerator Platforms with Chaos-on-Edge Hyperparameters

Anand Ravishankar[1], Santhi Natarajan[2] and Bharathi Malakreddy A[3]

[1]Amazon Web Services
[2]Department of Computer Science, Shiv Nadar University, Chennai
[3]Department of Artificial Intelligence and Machine Learning, BMSIT, Bangalore

Abstract—**Deep Neural Networks (DNNs) is an extremely attractive subset of computational models due to their remarkable ability to provide promising results for a wide variety of problems. However, the performance delivered by DNNs often overshadows the work done before training the network, which includes Network Architecture Search (NAS) and its suitability concerning the task. This paper presents a modified Genetic-NAS framework designed to prevent network stagnation and reduce training loss. The network hyperparameters are initialized in a "Chaos on Edge" region, preventing premature convergence through reverse biases. The Genetic-NAS and parameter space exploration process is co-evolved by applying genetic operators and subjugating them to layer-wise competition. The inherent parallelism offered by both the neural network and its genetic extension is exploited by deploying the model on a GPU which improves the throughput. the GPU device provides an acceleration of 8.4x with 92.9% of the workload placed on the GPU device for the text-based datasets. On average, the task of classifying an image-based dataset takes 3 GPU hours.**

Index Terms—**Deep Neural Networks, Network Architecture Search, Chaos on Edge Initialization, Co-evolutionary framework, Hardware Accelerations**

I. Introduction

Deep Neural Networks (DNNs) are computational structures consisting of multiple rudimentary units working in tandem to detect large-scale patterns. The steady rise in DNNs' popularity can be attributed to the creation of architectures such as AlexNet [1], ResNet [2], VGG [3] and LSTM [4]. The algorithms governing DNNs' functionalities have a strong underlying mathematical foundation, which has only been reinforced over the years with Improved Back-Propagation [5] and Conjugate Gradient Algorithm [6] being prime examples. Combined with hardware accelerators such as GPUs, ASICs, and FPGAs, parallel computation of DNNs has improved. The performance (*P*) of a DNN can be expressed as a function of three key components: Network Architecture (*NA*), Network Parameters (*NP*), and Knowledge Representation of the data provided (*K*).

$$P = f(NA, N_P, K)$$

Much emphasis is placed on improving the architecture and the rules governing a network's growth. With recent advances in hard-based accelerators, parallel frameworks such as DNNs and Genetic Programming (GP) operate on an increased throughput. Multiple execution threads co-operate to complete the workload quickly. The MPI and CUDA frameworks allow computing and data transfer I/O operations interleaving, resulting in optimized performance. The infatuation towards performance enhancement through hardware porting has often undermined the importance of network architecture selection and data representation. Given no prior knowledge, the Network Architecture Search (NAS) process is a trivial trial-and-error process. Moreover, given a network architecture and an optimized parameter set, the configuration stands good only for a given dataset and does not account for a runtime model with test data fluctuations. Modern NAS techniques can be subdivided into Reinforcement Learning (RL) based, such as [7] and [8], Evolutionary Techniques, such as [9], Bayesian Optimization [10] and Gradient-based Techniques [11]. RL techniques suffer a high computational cost for combing through the entire search space and adapting to the environment, while Gradient-based Optimization and Bayesian Optimization have time-consuming design processes. Applying a random search on network architecture would be time-consuming and computationally expensive as the number of configurations grows exponentially with even minor changes to the architecture. The shortcomings are listed below:

- Most systems are data set specific and need to be re-engineered for a new application.
- A new perturbation in the architecture space shall demand a fresh build of the whole system, rendering them outdated.
- Pre-defined mathematical functions such as ReLu are static and do not keep up with the evolving search space.

This paper presents an efficient genetic search method to construct a suitable network for a given dataset. In this work, we analyze the effect initializing the network hyperparameters in different domains has on the network performance and prevents premature convergence. Following this, the network is grown out over *G* generations. Once an error threshold is crossed, we begin to sparsify the network through vector-wise pruning. Furthermore, multiple iterators of the same base network are assigned a fitness value, and a layer-wise competition is introduced to select the best variant of the

anandravishankar12@gmail.com
santhinatarajan@snuchennai.edu.in
E.E@university.edu

base model. Finally, the selected architecture is ported onto a hardware accelerator to reduce inference complexity. Our main contributions are as follows:

- **HyperParameter Initialization**: The network hyperparameters are traditionally initialized to random values in a given range. However, recent research [12] [13] has shown that natural systems often operate on an "Edge of Chaos" (EoC). EoC forms the boundary between stable, orderly behavior and an unpredictable world of randomness. Systems functioning on EoC have displayed resilience to stagnation and offer increased system stability through anti-biases.
- **Pattern Aware Pruning**: Genetic programming is introduced to grow out the model to its optimal structure. To limit the growth of the transition matrix, pruning is applied to the genetic structure so formed. The pruning areas are determined by applying Back Propagation (BP) and Markov chain analysis to the transition matrix and the corresponding probabilistic states.
- **Layer-wise Competition**: For each layer, multiple instances are created, and the fitness of each instance is evaluated after every iteration. The Genetic-NAS process is shifted away from an accuracy-driven model, and emphasis is placed on individual contributions of each network variant.

Algorithm 1: Co-evolutionary NAS with parameter updation

```
Input: Training Set $D$, Fitness threshold $\lambda$, Base model $M$, Maximum
  Network Depth $max\_depth$, Maximum Nodes $max\_nodes$, Maximum
  Complexity $C$, Genetic Operators $G$
Result: Optimal Network Configuration is obtained
———————————————————— Initialize $I$ instances of
  Base Model with single block;
Initialize the hyperparameters in CoE;
for Each $n$ in $max\_nodes$ do
    Develop the network under constraint $C$ and $max\_depth$; for Each
      instance $i$ in $I$ do
        Store fitness scores in array $scores$[];
    end
end
for Each Instance $i$ in $I$ do
    Obtain Network configuration in an intermediate space through binary
      encoding; Obtain $I$ chromosomes; for Each potential solution
      chromosome in $I$ do
        Establish a many-to-one relation with $w$ parameter representations;
        Using $G$ evolve the chromosome;
    end
    Update $scores$[];
end
Translate the intermediate state to a binary evaluation tree;
for Each node $n$ in chromosome $i$ do
    Toggle the connection to the next layer using mutation;
    if Fitness score decreases then
        Apply Back propogation;
        Prune( Connection($i_n$) $->$ $((i+1)_n)$ );
    end
    else
        if Any chromosome $i$ decreases below $\lambda$ then
            break;
        end
        else
            continue;
        end
    end
end
```

II. Related Work

Recent efforts of integrating multi-disciplinary algorithms with NAS have given rise to a copious amount of techniques to automate the search process. As mentioned previously, the efforts can be broadly classified as Reinforcement Learning and Evolutionary Algorithms based on Gradient Optimization and Bayesian Optimization. RL-based techniques treat the entire search space as a learning environment with a parameter matrix of each layer's instance having feedback into the system for further refinement. Based on a pre-decided reward system, an external agent is employed to roam the search space and experience the environment. Evolutionary algorithms can be further sub-classified into Evolutionary Programming (EP) and Genetic Programming (GP). Both paradigms attempt to emulate Darwin's theory of evolution from different approaches with the help of genetic operators such as selection, crossover, and mutation. Gradient-based techniques utilize the gradient descent algorithm to test the goodness-of-fit of each layer's parameter.

- **RL based**: In [14] the authors generate high-performing CNN architectures for a given learning task using Q-learning with an ϵ-greedy exploration strategy and experience replay; In [15] the authors formulate the NAS as a Markov decision process which enables a more effective RL-based search. The samples generated by the previous policies are used efficiently for an off-policy GAN search algorithm.
- **Evolutionary based**: In [16] the authors construct a simplified supernet where all architectures are single paths that solve the weight co-adaption problem. Conduction of the search process in complex environments under multiple constraints is conducted seamlessly through uniform path sampling. In [17] evolved machine learning algorithms are created from scratch. They apply evolutionary concepts to top algorithms such as bilinear interactions, normalized gradients, and weight averages. In [18] a Genetic Algorithm (GA) is used to evolve ecological neural networks that can adapt to their changing environment. In [19] and [20], a GA is used on a fixed three-layer feedforward network to find the optimal mapping from the input to the hidden layer (i.e., the set of optimal hidden targets).
- **Gradient based**: Using Gradient descent offers a big advantage in terms of reduced GPU-hours as shown by [21]. The entire search space is represented in directed acyclic graphs, each one of which constitutes a possible network architecture. A differential sampler is then trained through gradient descent which optimizes the learning process. The resultant network took only 4 GPU hours with a test error of 2.82%. Another method involved optimizing the network configuration through stochastic relaxation as shown by [22]. The authors apply the gradient descent technique with an adaptive step-size method for low-cost computations.

III. Co-Evolutionary NAS with network configuration optimization

In this section, we discuss the modifications applied to the Genetic-NAS process along with the co-evolutionary growth of the architecture and the associated parameters. We begin our discussion with hyperparameter initialization. We briefly discuss the implications of developing systems in the CoE domain rather than the randomized domain. Following this, we discuss the core Genetic-NAS process and the corresponding GPU partitioning of the training phase. The goal of developing an end-to-end framework consists of three main components, each focusing on one aspect of optimizing the base model's performance. An additional component in the form of GPU implementation is added to speed up the Genetic-NAS process.

The end-to-end framework is described in Algorithm 1 and discussed in depth in the following subsections.

A. *Hyperparameter Initialization*

The impact of chaos in discrete systems was studied extensively by Langton and Kauffman [23] [24]. The introduction of chaotic variables into genetic programming occurs not as a low dimensional determinstic choas but rather in a phase where some interesting features begin to appear.

We define a parameter λ that varies incrementally from complete chaos to a fixed state. At a critical value of λ, λ_c, our hyperparameters exhibit longer, complex, aperiodic but non-random patterns of connectivity. Let n and d be the number of hyperparameters and the range of a particular hyperparameter. The corresponding number of hyperparameter states will be given by Equation 1.

$$T = n^d \tag{1}$$

If there are q latent states, then λ is given by Equation 2.

$$\lambda = \frac{T - q}{T} \tag{2}$$

Figure 2 shows the variant initialized in CoE achieves a lower error as compared to random initialization for 200 epochs.

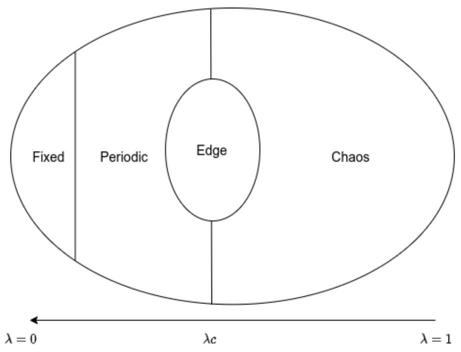


Fig. 1. Different Domains for Initialization

B. *Co-Evolutionary NAS*

After the base model and the hyperparameters are finalized, we begin to optimize the network architecture and the parameter set. The general flow of this growth for a sample CNN is shown in Figure 3.

A co-dependent structure for optimizing the NAS process and the weight selection is employed using structured chromosomes. For computational purposes, an intermediate state of binary matrix encoding is introduced. An n-dimensional matrix scheme has been implemented to represent the DNN structures concisely. It encodes regularities in the connectivity matrix defining the network structure. Another added advantage of the binary representation is the translation of genetic operators as structural changes in the network, which depend on the chromosome layer they are acting on. A many-to-one mapping is employed between the architectural representation and the weight representation to verify the goodness-of-fit of the model as shown in Figure 4.

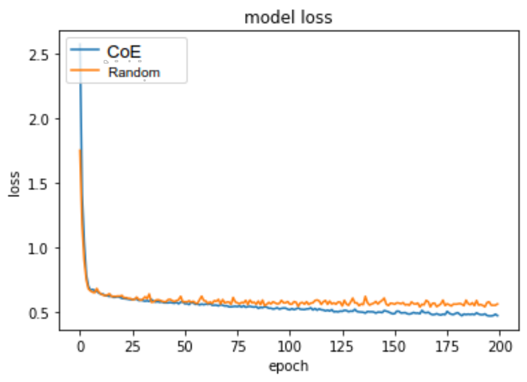


Fig. 2. Loss for variants initialized randomly and in CoE for 200 epochs

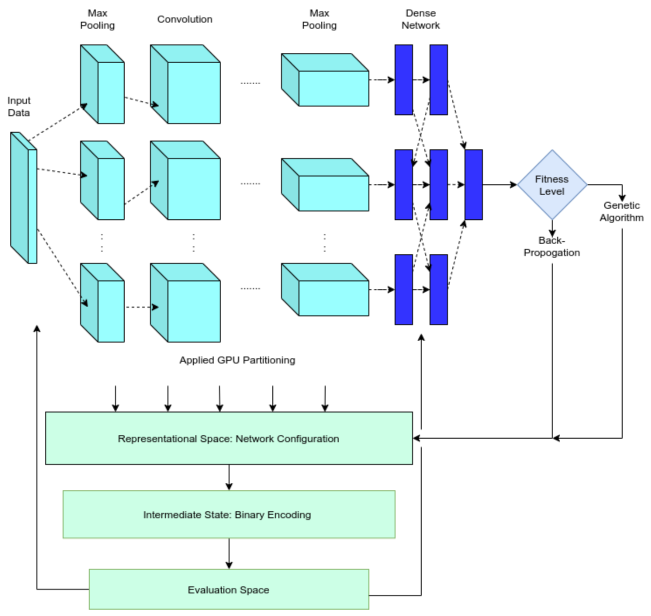


Fig. 3. Genetic Programming based NAS

*1) **Pruning**:* In the evaluation stage of the encoding system, the generated networks are first pruned before they are trained, using the Back Propagation (BP) module. Pruning removes neurons and the corresponding links that have no incoming or outgoing connections. It does this recursively until all such neurons have been removed. BP training is performed till the error threshold is crossed since network convergence is dependent on the network's configuration. The optimal cardinality depends on the dataset provided, and the train-test split set used. The BP module uses a gradient descent weight updating rule with a momentum term. The module uses a per-pattern weight update mechanism, meaning that the weights are updated after every iteration of the training phase.

The fitness function used to evaluate the goodness-of-fit of the network instance is given by Equation 3

$$F = \frac{\hat{Y} - Y}{\sum_{i=0}^{N} O_{\mathrm{i}}} + \alpha \frac{C}{C_{\max}} + \mu P \tag{3}$$

where $\hat{Y}$ is the target variable, Y is the predicted output, O_{i} is the number of output classes, C is the network complexity, $C_{\max}$ is the maximum complexity, P is the fraction of the network pruned, α is the learning rate and β is momentum. Note that since the fitness function is directly proportional to the loss, the problem of optimizing the neural network is a minimization problem rather than the usual maximization problem.

*2) **Markov Chain analysis**:* The most common methods for analyzing genetic structures include the Schema theorem, Price's theorem, and building block hypothesis. The methods mentioned provide a good approximation of the genetic model and have been used often. Markov Chain analysis, however, replicates an exact model of the genetic structure in a state-space diagram. The models so formed provide useful insight into the effect of genetic operators on the transition matrix.

Since binary encoding was employed, the chromosomes are treated as binary strings, with each gene representing "no connection" or 0 and "connection" or 1. If N chromosomes of length l are generated, the number of possible states is $N + 1$. State i is referred to as the state with exactly i ones and $(l - i)$ zeros. The operation of the genetic algorithm is now defined by a $(l+1)*(l+1)$ transition matrix $P[i, j]$ that maps the current state i to the next state j . The probability of a transition from state i to state j is given by one entry in the matrix $p(ij)$.

If f_0 is the fitness of the chromosome of length l having i zeros when the connection is severed, and f_1 is the fitness of the chromosome when the connection is retained, the probability of the connection surviving is given by Equation 4.

$$p_1 = \frac{i * f_1}{i * f_1 + (l - i) * f_0} \tag{4}$$

And the probability of connection not surviving is given by Equation 5.

$$p_0 = \frac{(l - i) * f_0}{i * f_1 + (l - i) * f_0} \tag{5}$$

Hence the probability of transition from state i to j can be written as:

$$p(i, j) = \binom{l}{j} (p_1)^{\mathrm{j}} (p_0)^{\mathrm{l-j}} \tag{6}$$

Equation 6 completely describes the transition matrix with no preference for a state(i.e., a population) with all-ones or a state with all-zeros, and the equation reduces to the one for pure genetic drift.

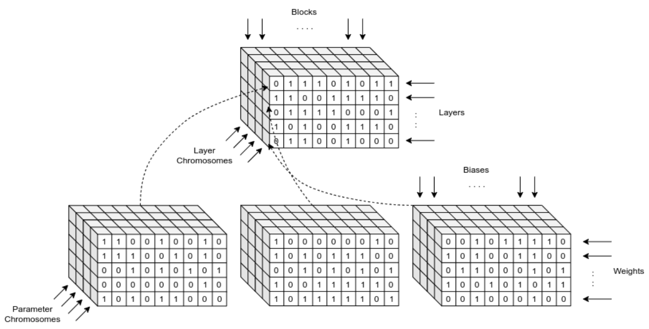


Fig. 4. Many-to-one relation between NAS and weight representation

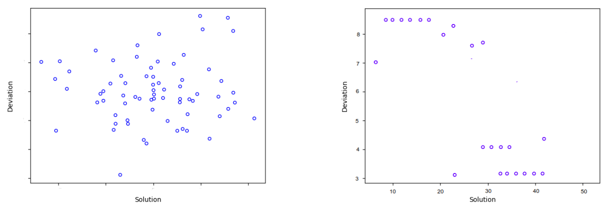
IV. Experimental Results

In this section, we describe the experimental setup and follow it up with a discussion of the results. We have implemented the GP algorithm for the NAS process. We have used evolutionary operators, along with BP, for parameter setting and network convergence. The rig setup for our experiment consisted of an Intel(R) Core(TM) Xeon Silver 4208 CPU@2.50GHz with 32GB RAM. We developed the DNN through Tensorflow v2.3 and deployed it on an HPC system with a PowerEdge R740 motherboard and 1 NVIDIA Tesla V100-PCIe GPU card (5120 CUDA cores with a peak throughput of 112 TFLOPS). We conducted the experiments on six public datasets: MNIST digit classification [25], CIFAR-10, CIFAR-100 [26], Boston Housing prices dataset [27], IMDB movie review sentiment dataset [28] and Reuters newswire dataset [29]. The network settings are described in Table I.

1) Hyperparameter Initialization: We conducted the initial experiments on text-based datasets i.e, [28], [29] and [27]. The hyperparameters of the base model belonged in one of the two domains: Complete Chaos and CoE. We created ten variants of the base model, i.e., Feedforward network, for both the domains. Figures 5 and 6 display the effect of hyperparameter initialization in different domains. The models were grown over 50 generations. When created randomly, the hyperparameters are forced into stagnation, beyond which no genetic drift is observed. However, when the hyperparameters are initialized in CoE, they display long-standing randomness. λ_{c} was evaluated to 0.786 for text-based datasets and 0.881 for image-based datasets.

TABLE I
NETWORK SETTINGS

Parameter	Setting
Learning Rate	X ∈ [0.0001-1.0]
Momentum	X ∈ [0.0-0.9]
Back Propogation Cycles	<50
Encoding	Binary
Selection	Elitism
Mutation	Binary Inversion
Crossover	Single Point
Mutation Rate	X ∈ [0.0-1.0]
Crossover Rate	X ∈ [0.001-1.0]
Maximum Depth	5
Maximum Nodes	1024
l	31
N	10
Number of Generations	10
Number of epochs	10
Replacement	Unconditional

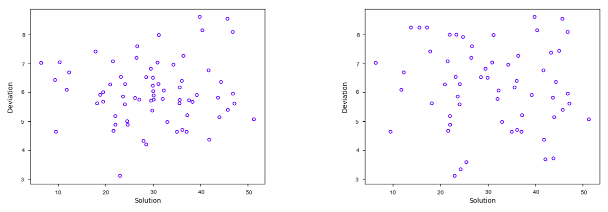


(a) Randomized Initial Network Hyperparameters Deviation against Ground Truth

(b) Randomized Network Hyperparameters Deviation after 50 iterations against Ground Truth

Fig. 5.

2) Co-Evolved Genetic NAS: Both the structural component of the chromosomes and the connections use binary encoding. The coding of the structural part is, in general non-homogeneous, each having its gene length. In our present work, we consider only homogeneous chromosomes because implementing non-homogeneous chromosomes in core genetic programming proved challenging. We will try to implement heterogeneous encoding in our future works. Table II shows the average number of generations it takes the CoE variant to converge for different levels of sparsity. The variants are evolved over 200 generations, and the model with the best fitness for each dataset is also shown in Table II. Note that the higher the fitness value, the more generalized the network is. As we can see, the variant generalizes well for all datasets.



(a) CoE Initial Network Hyperparameters Deviation against Ground Truth

(b) CoE Network Hyperparameters Deviation after 50 iterations against Ground Truth

Fig. 6.

3) Impact of Hardware Accelerators: Due to the parallel nature of DNNs, implementing them on GPUs reduces the average step time considerably. Moreover, We can apply genetic operations such as crossover and mutation in parallel as the operations are independent of the data involved. The placement of Tensorflow operations on a GPU device provides an acceleration of 8.4x, with 92.9% of the workload being placed on the GPU device. On average, the task of classifying an image-based dataset takes 3 GPU hours. The reduced time frame enables an increased throughput while accelerating the design space exploration process compared to a sequential search.

V. CONCLUSION AND FUTURE WORK

The proposed framework initializes the network hyper-parameters in the CoE region. Due to this, the network avoids stagnation and premature convergence. The model is then co-evolved in terms of the network architecture and the parameters defining its configuration. The evolution is done with the help of genetic programming, which translates the actual configuration into a binary space representation. Genetic modifiers such as crossover, mutation, and selection are applied, and the model is continuously graded against a fitness function. Backpropagation is applied to prune the network and reduce the growth of the binary encodings.

The genetic components of our framework offer a wide array of advantages such as a decrease in computational cost, avoidance of local optima and does not mandate an absolute representation of the network. However, due to the binary encoding system, the complexity increases exponentially as the size of the problem increases. This issue is, to a great extent, addressed by the introduction of a hardware accelerator. We can make additional improvements to this project in terms of finer filters, an amalgamation of co-evolution with RL for improved performance, and new baseline architectures.

REFERENCES

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In NIPS.
[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In CVPR.
[3] Karen Simonyan and Andrew Zisserman. 2015. Very Deep Convolutional Networks for Large-scale Image Recognition. In ICLR.
[4] Sepp Hochreiter; Jürgen Schmidhuber (1997). "Long short-term memory". Neural Computation.
[5] Nawi N.M., Ransing R.S., Salleh M.N.M., Ghazali R., Hamid N.A. (2010) An Improved Back Propagation Neural Network Algorithm on Classification Problems. In: Zhang Y., Cuzzocrea A., Ma J., Chung K., Arslan T., Song X. (eds) Database Theory and Application, Bio-Science and Bio-Technology. BSBT 2010, DTA 2010. Communications in Computer and Information Science, vol 118. Springer, Berlin, Heidelberg.
[6] B. Wang, B. Yang, J. Sheng, M. Chen and G. He, "An Improved Neural Network Algorithm and its Application in Sinter Cost Prediction," 2009 Second International Workshop on Knowledge Discovery and Data Mining, Moscow, 2009, pp. 112-115, doi: 10.1109/WKDD.2009.180.

TABLE II
CO-EVOLVED NAS RESULTS

Dataset	Average Generations for Convergence (Pruning = 0.5)	Average Generations for Convergence (Pruning = 0.7)	Best Fitness after 200 Generations
MNIST	124.84	105.42	96.48
CIFAR-10	264.87	218.21	87.56
CIFAR-100	709.93	624.78	91.83
Boston Housing	84.5	62.14	77.61
IMDB Ratings	78.94	77.91	81.49
Reuters Dataset	130.47	109.86	92.87

[7] Barret Zoph, Quoc V. Le, Neural Architecture Search with Reinforcement Learning, arXiv:1611.01578

[8] H. Liu and C. Zhang, "Reinforcement Learning based Neural Architecture Search for Audio Tagging," 2020 International Joint Conference on Neural Networks (IJCNN), Glasgow, United Kingdom, 2020, pp. 1-8, doi: 10.1109/IJCNN48605.2020.9207530.

[9] Clifford Broni-Bediako, Yuki Murata, Luiz Henrique Mormille, Masayasu Atsumi, Evolutionary NAS with Gene Expression Programming of Cellular Encoding, arXiv:2005.13110

[10] L. Ma, J. Cui and B. Yang, "Deep Neural Architecture Search with Deep Graph Bayesian Optimization," 2019 IEEE/WIC/ACM International Conference on Web Intelligence (WI), 2019, pp. 500-507.

[11] Liam Li, Mikhail Khodak, Maria-Florina Balcan, Ameet Talwalkar, Geometry-Aware Gradient Algorithms for Neural Architecture Search, arXiv:2004.07802

[12] Bassett, D., Meyer-Lindenberg, A., Achard, S., Duke, T., Bullmore, E.: Adaptive reconfiguration of fractal small-world human brain functional networks. The National Academy of Sciences of the USA 103(51), 19518–19523 (2006)

[13] Kitzbichler, M., Smith, M., Christensen, S., Bullmore, E.: Broadband Criticality of Human Brain Network Synchronization. PLoS Comput. Biol. 5(3), e1000314 (2009)

[14] Baker B, Gupta O, Naik N, et al. Designing neural network architectures using reinforcement learning[J]. arXiv preprint arXiv:1611.02167, 2016

[15] Yuan Tian, Qin Wang, Zhiwu Huang, Wen Li, Dengxin Dai1, Minghao Yang, Jun Wang, and Olga Fink, Off-Policy Reinforcement Learning for Efficientand Effective GAN Architecture Search. arXiv:2007.09180

[16] Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, Jian Sun, Single Path One-Shot Neural Architecture Search with Uniform Sampling, arXiv:1904.00420

[17] Esteban Real, Chen Liang, David R. So, Quoc V. Le, AutoML-Zero: Evolving Machine Learning Algorithms From Scratch, arXiv:2003.03384

[18] Lund, H.H. and Parisi, D., "Simulations with an Evolvable Fitness Formula" Technical Report PCIA-1-94, C.N.R., Rome, 1994.

[19] Hassoun, M.H., Fundamentals of Artificial Neural Networks, MIT Press, 1995.

[20] Munro, P.W., "Genetic Search for Optimal Representations in Neural Networks", International Conference on Artificial Neural Nets and Genetic Algorithms (ANNGA93), Innsbruck, Austria, pp. 628-634, 1993.

[21] Xuanyi Dong, Yi Yang Searching for A Robust Neural Architecture in Four GPU Hours. arXiv:1910.04465

[22] Youhei Akimoto, Shinichi Shirakawa, Nozomu Yoshinari, Kento Uchida, Shota Saito, Kouhei Nishida, Adaptive Stochastic Natural Gradient Method for One-Shot Neural Architecture Search, arXiv:1905.08537

[23] Kauffman, S.: Antichaos and Adaptation. Scientific America, 78–84 (August 1991)

[24] Kauffman, S.: At Home in the Universe. Oxford University Press, Oxford (1995)

[25] LeCun, Yann and Cortes, Corinna. "MNIST handwritten digit database." (2010):

[26] Learning Multiple Layers of Features from Tiny Images, Alex Krizhevsky, 2009.

[27] Harrison, D. and Rubinfeld, D.L. Hedonic prices and the demand for clean air, J. Environ. Economics and Management, vol.5, 81-102, 1978.

[28] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. (2011). Learning Word Vectors for Sentiment Analysis. The 49th Annual Meeting of the Association for Computational Linguistics (ACL 2011).

[29] Sam Dobbins, Mike Topliss, Steve Weinstein Reuters-21578, 1987