Nerual Architecture Search with genetic algorithms and DARTS

Thomas Reolon - *thomas.reolon.it@gmail.com*

***Abstract*—Given the recent success of deep learning, new fields that try to improve neural networks are emerging, Neural Architecture Search (NAS) is one of them. We propose a system that can autonomously find good NN architectures using a genetic algorithm. To speedup architectures' evaluations we use heuristic functions that do not require training the NN, and a variation of DARTS.**

***Index Terms*—NAS, NTK, DARTS, GA, neuroevolution**

I. Introduction

IT is possible to embed some prior knowledge in the architecture of a neural network (NN), for example Adam Gaier et al. [1] showed that we can build weight agnostic neural networks, which can perform tasks even without tuning their weights. Moreover many breakthroughs in deep learning came from architectural innovations, eg. the use of CNN in Alexnet [2] (2012), skip-connections in Resnet (2016) [3], self-attention in Transformers (2017) [4], ...

Even if NAS has just lately become a popular topic, some very interesting approaches like NEAT [5] had already been proposed in the past. This approach uses a genetic algorithm (GA) to evolve the architecture and the weights of a NN. More recently a variation called CoDeepNEAT [6] suggested a way to evolve architectures with more parameters (in NEAT you evolve weights and connections, in CoDeepNEAT you evolve layers and connections between them), moreover, they evolve cells, which can be seen as small NN, and divide them into species; once the evolution phase is complete, they build the final architecture by combining these cells. In this project, we use the same cell-based approach.

Even if the literature contains many BIO-inspired algorithms to perform neural architecture search (eg. deepswarm [7]), the most popular approaches use other techniques such as weights sharing and network morphisms [8][9][10]. Among these approaches a very simple ed effective method is DARTS [11], which can be summarized as follows: we use backpropagation to learn a parameter called alpha, which selects which connection (dense, cnn, max-pooling, ...) we should keep in our final architecture.

One of the biggest issues in using GA for NAS was how to evaluate an architecture. A commonly used fitness function was the accuracy obtained by the NN after some training epochs, but even if this approach produced very good architectures it required enormous quantity of computing power (eg. AmoebanetA used weeks of GPU computing power [12]). Given some recently discovered properties of NN, some methods to evaluate architectures without training them were proposed [13] [14], we base our approach upon these methods.

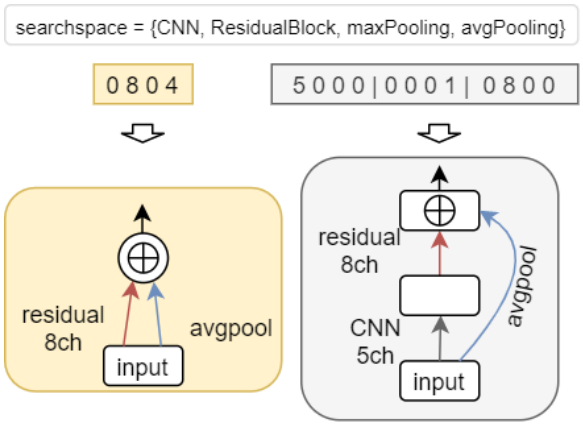

Fig. 1. genotypes encoded as strings and their phenotype. The number of channels indicate if that connection is used or not. the bars "—" delimitate blocks, each block has a different combination of input output depth

In particular, we adopt the heuristic functions to replace the old accuracy fitness function.

Section II presents our method, Section III shows the obtained performances (plot of the fitness and accuracy function) and section IV contains some thoughts on this subject.

II. Method & Implementation

NOW we present the most salient elements of the algorithm, then we explain how an iteration/epoch is run.

GENOTYPE

The genotype represents the design of a cell (that is: which layers are used, how they are connected and how many parameters they have). To encode the cells we used a similar approach to the one used in DARTS; we preferred this approach over the encoding used in NEAT because there were more code-examples available online (for pytorch). *Fig 1* shows how the genotype is encoded in practice.

PHENOTYPE

Pytorch was chosen as deep learning framework to build the phenotypes; we took inspiration on how to dynamically build pytorch's NN from the TENAS project [14].

FITNESS FUNCTIONS

We have three functions to minimize:

- the Neural Tangent Kernel score (implementation taken from TENAS project with some changes to avoid running out of GPU-RAM)
- the Linear Region Score (from [13])
- the Gradient Correlation score created following the ideas explained in Yannic Kilcher's videos [15][16]

You can find more information about these functions in Appendix A.

CROSSOVER

We use a simple crossover strategy: given that the genotype is encoded as a string, the $i-th$ character of the offspring will be randomly selected between the $i-th$ character of the two parents.

MUTATION

The main mutation strategy consists in adding some new channels in the architecture, there are some parameters which decide how to do these mutation and are learned during the iterations (global evolution strategy). There are some other types of mutations, eg. swapping blocks, reduce the number of channels, ...

SELECTION

Once we have the 3 scores for each offspring, we select the architectures that were not dominated by others. If there are too many offsprings after this selection we further reduce their numbers with the method proposed in [14]. This is a $(\mu + \lambda)$ GA because parents can survive multiple generations.

DARTS STEP

It is probable that minimizing these three scores will introduce some bias in the found architectures. To minimize this bias, once every three generation, we use another approach to evaluate/select architectures. Instead of using the three heuristic functions and the Pareto dominance selection, we create some tournaments between the offsprings where, only one offspring per tournament will survive. This tournament is created adopting the DARTS method $Fig2$, but instead of selecting connections we select cells designs.

The pseudo code below briefly shows how a generation is computed

Once the algorithm completes the final epoch, we store the final population on a file. To obtain the final architecture, we concatenate the cells as in $Fig.3$ and add some final pooling/dense layers.

III. RESULTS

BEFORE presenting the results obtained with the GA, we want to show a sanity check we performed to verify that the heuristic function were effective. As we can see, Fig. 4 shows that the scores for famous architectures (eg. resnet, VGG, ...) are very close to the origin, while randomly generated architectures get very poor scores with an high variance.

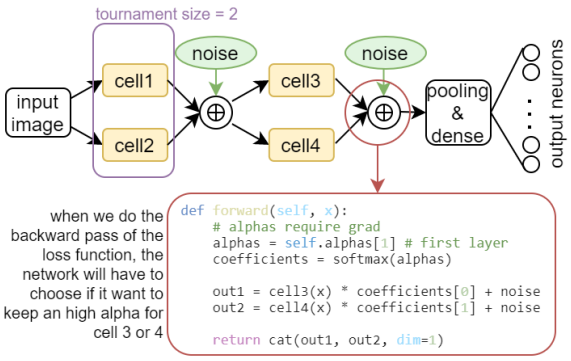

Fig. 2. tournament selection using DARTS. The network contains many bifurcations, after training the network with gradient descent, it will choose which cell to keep for each bifurcation

Algorithm 1: Do One Generation

```
Input: initial_population, use_darts_tournament, data
       set, tournament_size
Output: new_population
offsprings = mutation_and_crossover(population)
pop_size = length(population)
if use_darts_tournament then
    dart_network = network(offsprings,
     depth=pop_size)
    train(dart_network)
    new_population = get_winners(dart_network)
else
    offsprings = get_3_scores(offsprings, dataset)
    sort_by_TENASscore(offsprings)
    sort_by_paretodominance(offsprings)
    new_population = offsprings[:pop_size]
end
```

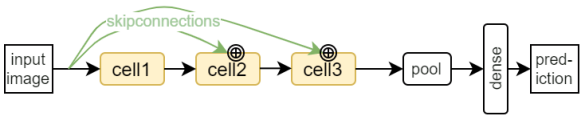

Fig. 3. final architecture obtained combining multiple cells. It takes as input an image and returns the predicted score for each class (CIFAR-10 has 10 classes)

We now report the results we obtained running the GA for some hours. The plot in Fig. 6 shows the evolution of the "best" offsprings of each generation, while Fig. 5 shows the best score for each fitness function for each generation, as we can see the three heuristics (that we want to minimize) decrease at each generation.

Even if there are some benchmarks to evaluate NAS architectures on CIFAR-10 (eg. NAS-Bench-201), the architectures created by this approach fall off these benchmarks. For this reason we had to compute the accuracy by training them from scratch. To compute the accuracy for the last epoch we build a NN which is the concatenations of the evolved cells (Fig. 3).

As we can see, the accuracy in the last epoch reached 70% while a resnet18 trained for the same amount of time on a

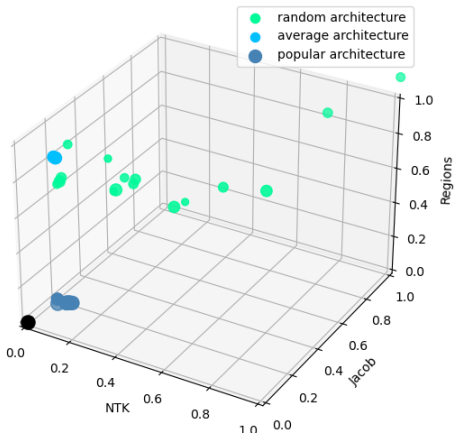

Fig. 4. scores obtained by different architectures: the closer to the origin (black sphere) the better. As we can see, popular architectures like resnet/VGG perform a lot better than small handcrafted CNN or NN created from randomly mutated genotypes. We can notice that the axis have been resized between [0,1] comparing this plot with the one in Fig. 5, this was needed to cutoff some outliers (some random architectures had such bad scores that the plot was deformed)

GTX-1080 GPU reached 82%. Even if the results are a bit worse, this result is still interesting, because our architecture uses almost 20 times less parameters.

IV. Conclusions

THIS work is built upon the works of Chen et al. [14] and Mellor et al. [13] and tries to explore the potential of applying genetic algorithms to NAS. Potentially there could still be months of work to make improvements, so the results should be taken lightly.

There were 2 main kinds of problems encountered during this project: technical (eg. GPU-memory handling, creating NN dynamically, ...) and practical (at the beginning it was not working very well due to bad choices of possible mutations, search space (skip-connection are a bit bugged as stated in [17] and [14] too)).

A nice property that many NAS implemented projects have is the capability to change the connections (the bricks of the cells, eg. CNN, avgPooling, ...). This system makes no difference and allows you to create your own connections (as long as they offer the same API of a CNN), this could allow building very interesting experiments (eg. introducing attention or other types of layers). Anyway, this capability is also offered by simpler algorithms such as DARTS or TENAS.

There still are margin of improvements: better heuristic functions, improve mutation and crossover, but we are satisfied with our work. Probably in some months some new papers on this topic will be released.

Appendix A
Heuristic-Based Fitness Functions

Neural Tangent Kernel

How To Compute: We take N samples, and for each of them we do the forward and the backward pass. We then store the gradients in a matrix M, where M_{ij} is the gradient of the parameter j for sample i (a NN usually has many output neurons, so a better description of the gradients would be: "the flattened jacobian matrix of size: (numberofparameters, number of output neurons)"). We then compute the covariance between the samples' gradients as $C = M \cdot M^T$. Finally we compute the eigenvalues and return as final score the ratio between the biggest and the smallest one.

Intuition: A neural network has a good potential if there is a lot of variance between the gradients generated by different samples. If the ratio between the 2 eigenvalues is small, we don't have a clear principal component that describe how they are distributed, thus we have a high entropy/variance. More: [14].

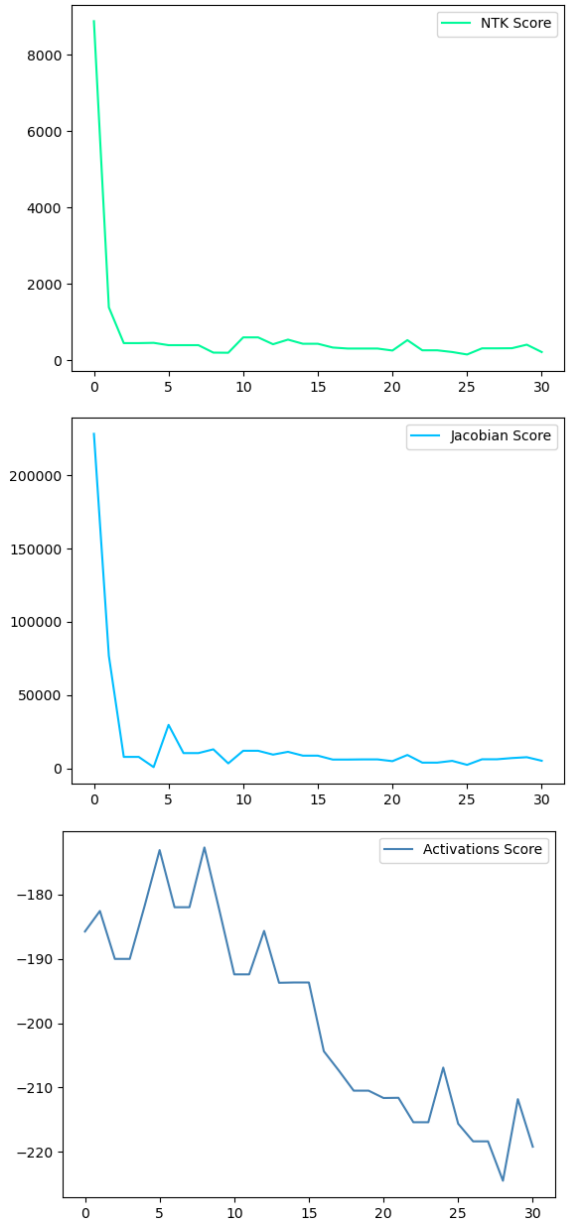

Fig. 5. the evolution of the three heuristic functions. The oscillations of the scores are caused by an aging factor (even if is a $(\mu + \lambda)$ GA, it is possible for a parent to die if it is unlucky)

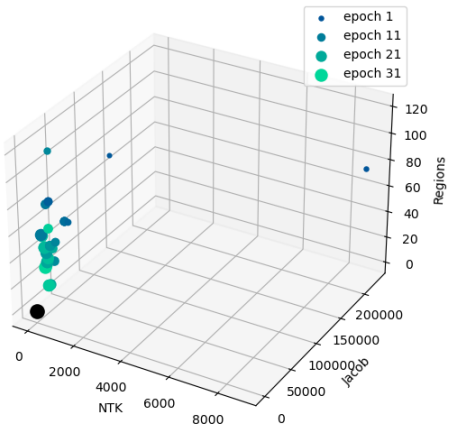

Fig. 6. evolution of the best offsprings: in the beginning (blue) the offsprings are far from the origin, as evolution proceeds, they get closer to the origin (green).

Activation Score

Intuition: A neural network would appreciate if the activations of different inputs were different between each other. This score measures these dissimilarities by computing a matrix M where M_{ij} is "how many activations have the same score between sample i and sample j". If M is a diagonal matrix the determinant is maximized and the activation are very different between each other ($score = -log(determinant)$). More: [13].

Third Score

Intuition: This score is very similar to NTK, but we also take into account how similar the two input images are: the more different the more they are important.

APPENDIX B

An implementation of this algorithm can be found online at: https://github.com/thomasreolon/Evolutive-NAS

REFERENCES

[1] A. Gaier and D. Ha, "Weight agnostic neural networks," *arXiv preprint arXiv:1906.04358*, 2019.
[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012.
[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.
[4] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *arXiv preprint arXiv:1706.03762*, 2017.
[5] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary computation*, vol. 10, no. 2, pp. 99–127, 2002.
[6] R. Miikkulainen, J. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, H. Shahrzad, A. Navruzyan, N. Duffy *et al.*, "Evolving deep neural networks," in *Artificial intelligence in the age of neural networks and brain computing*. Elsevier, 2019, pp. 293–312.
[7] E. Byla and W. Pang, "Deepswarm: Optimising convolutional neural networks using swarm intelligence," *arXiv preprint arXiv:1905.07350*, 2019.
[8] T. Elsken, J. H. Metzen, F. Hutter *et al.*, "Neural architecture search: A survey." *J. Mach. Learn. Res.*, vol. 20, no. 55, pp. 1–21, 2019.
[9] F. Hutter. (2020) Directions in ml: "neural architecture search: Coming of age". Microsoft Research. [Online]. Available: https://www.microsoft.com/en-us/research/video/directions-in-ml-neural-architecture-search-coming-of-age/
[10] A. Talwalkar. (2020) Algorithmic foundations of neural architecture search. Microsoft Research. [Online]. Available: https://www.microsoft.com/en-us/research/video/algorithmic-foundations-of-neural-architecture-search/
[11] H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable architecture search," *arXiv preprint arXiv:1806.09055*, 2018.
[12] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *Proceedings of the aaai conference on artificial intelligence*, vol. 33, no. 01, 2019, pp. 4780–4789.
[13] J. Mellor, J. Turner, A. Storkey, and E. J. Crowley, "Neural architecture search without training," *arXiv preprint arXiv:2006.04647*, 2020.
[14] W. Chen, X. Gong, and Z. Wang, "Neural architecture search on imagenet in four gpu hours: A theoretically inspired perspective," *arXiv preprint arXiv:2102.11535*, 2021.
[15] Y. Kilcher, "Neural architecture search without training (paper explained)," 2020. [Online]. Available: https://www.youtube.com/watch?v=a6v92P0EbJc
[16] ——, "Deep networks are kernel machines (paper explained)," 2021. [Online]. Available: https://www.youtube.com/watch?v=ahRPdiCop3E
[17] P. Wang, Y. Li, and N. Vasconcelos, "Rethinking and improving the robustness of image style transfer," *arXiv preprint arXiv:2104.05623*, 2021.