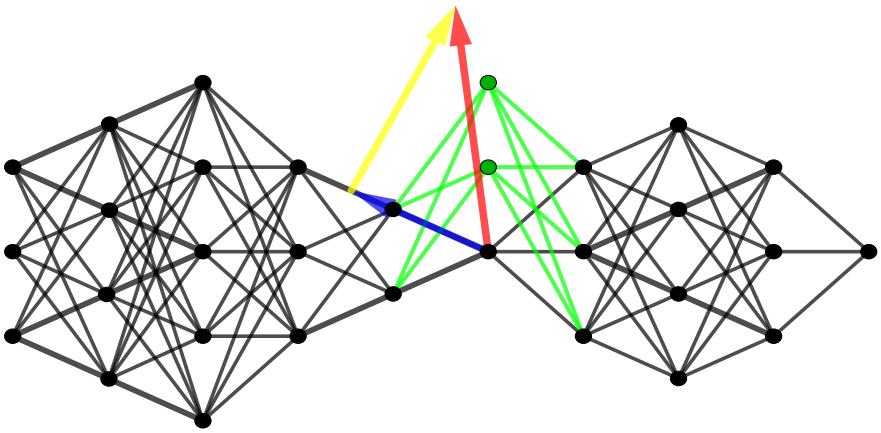
Neural Architecture Growth for Frugal Learning



1 Intership context

Research lab: INRIA TAU team (joint team between INRIA, CNRS and LISN of Université Paris-Saclay)
Location: LISN (building 660 "Digiteo", at Université Paris-Saclay)
Supervision team: Guillaume Charpiat, (guillaume.charpiat@inria.fr), Sylvain Chevallier, Stella Douka, François Landes, Stephane Rivaud and Théo Rudkiewicz
Funding: European project MANOLO

2 Problem statement

Thematic context Deep learning has shown impressive, highly-mediatized results on various applications (Go game, StarCraft, translation, object detection in images, high-resolution image generation, text generation...), obtained at the cost of training huge neural network architectures, which therefore also takes time and money (for instance, GPT-3 has 10^{11} parameters and might cost millions of dollars to be trained), both at training and exploitation times. Frugal learning, on the opposite, consists in training with as few samples or as little computational power (Green AI) as possible. We will focus on the latter here.

Large vs. small One advantage of having many neurons per layer is that it is known (experimentally and theoretically [GJS+20]) to facilitate optimization during training, thus yielding better results. However, trained neural networks show high internal redundancy, and various techniques have been developed to squeeze them into smaller networks with comparable accuracy. For instance [LUW17] manages to divide by 100 the number of neurons in order to run online object recognition in videos on a smartphone. On the other extreme, training and applying "tiny" models

(with "only" 10^5 parameters) will be much faster (as each test of the network is of much smaller complexity) but they might suffer from a lack of expressivity, preventing them from fitting data accurately.

Change of paradigm In our work (soon to be) published in TMLR [VRCC24], we propose to start with the simplest possible neural network (i.e., one neuron, a.k.a., linear regression) and make the network grow according to the information brought by the backpropagation pass. Such information can indeed be used to go beyond usual limitations in small network training, to explicitly tackle potential optimization and expressivity issues. To do this, we locate precisely, while training, the learning bottlenecks of a neural network, i.e., the layers which lack expressive power, in order to boost them by adding neurons or new layers appropriately where and when it is needed. With such an iterative architecture refinement scheme, important gains in architecture search (auto-DL) are expected. Indeed current Neural Architecture Search methods consist in trying many different architectures, while with our approach a single training progressively grows the architecture.

3 State of the art

Work done on this topic in our team so far A PhD student (Manon Verbockhaven) has already formalized mathematically the concepts required to spot and fix expressivity bottlenecks. She has also implemented layer growth of fixed convolutional and fully-connected architectures. More details concerning this methodology can be found in [VRCC24]. A master 2 intern (Barbara Hajdarevic) has extended this work to layer graph growth, i.e. adding new layers to the computational graph. This is a necessary step to unleash the power of the approach and to check how it performs. Currently, 2 PhD students (Stella Douka and Théo Rudkiewicz) and a Post-Doctoral researcher (Stéphane Rivaud) are also working on the topic. Our networks are growing and so is our team.

State of the art and other methods The main approaches to full neural network architecture optimization are based on automatic hyper-parameter tuning (auto-DL), but they are computationally extremely demanding, as they run many tries on many architecture variations. Some approaches incorporate architecture flexibility in their design [LSY19]; they can however only suppress connections between existing blocks, or suppress blocks, but not add new ones. Only a few approaches try to grow architectures (such as [MRLW22]); unfortunately, they are most often based on ad hoc criteria which are not mathematically justified.

4 Scientific proposal

We have many open axes of research that we want to tackle. Therefore, we offer multiple internship positions. Our current work offers a good overall method for growing neural networks, but some crucial part of the protocol could be studied to potentially largely improve our method. Our method allows finding good initialization of network extension and valuable information about the impact of those extension. Among possible topics, keeping in mind that more are available depending on the candidate's profile:

- **When to add neurons? Training/updating compromise** Using natural gradient and solving exactly the linear regression problem at each neuron addition, one may expect that no additional training (by gradient descent) is needed. In practice, we use a first-order method: we solve exactly to obtain the ideal direction, and move by a small step, controlled by an *amplitude factor*. We then complete training with a usual gradient descent. We want to **study the trade-off** between training by **gradient descent** and **updating the architecture** (neuron addition), which is implicitly controlled by the amplitude factor (and other choices).

- **Robustness of statistical estimators.** The expressivity bottleneck is estimated using a mini batch of data of a given size. We want to study the **statistical robustness** of this estimation and provide a theoretical bound. We also want to **study (theoretically and/or empirically)** how to best choose the ratio between the batch size used for gradient descent and the statistical batch size used for estimating the expressivity bottleneck.

- **Where to add neurons?** If we know where we want to add neurons, we can find a good initialization and the first-order impact of the loss. We could choose to add new neurons where there will be the most efficient at first order but is this greedy strategy good? Also, how to go beyond first-order approximations, as needed when creating new layers?

  - **Algorithmics for expressivity bottlenecks localization**. Backpropagation properties might be better exploited to design new algorithms to identify expressivity bottlenecks faster. Ideally, predict the location where the expressivity bottleneck is maximized **without having to calculate it**.
  - **Reinforcement Learning.** Define a neural network state descriptor to efficiently describe the state of the model (size, expressivity bottlenecks, architecture, complexity, performance, etc.). This can be used by Reinforcement Learning algorithms to define a growth strategy.

Note that we already have a well-defined **theoretical framework** [VRCC24], a **good codebase** (`https://gitlab.inria.fr/mverbock/tinypub`, currently being refactored by the team) and have defined **several experimental test cases**.

5 Expected results

Depending on the subject, the results can be theoretical, empirical or both.
A theoretical could provide new guarantees to a used algorithm. It can also give useful insight on the growing process which could later be used to improve the method.
For empirical results, the method development can be tested on various standard image classifications benchmarks (MNIST, CIFAR, ImageNet) and should be compared with existing method from the team and the literature. In addition the experiments should be reproducible and the new code should be documented to be reusable.

6 Expected skills

The required skills are the central ones for any ML researcher both on the theoretical and technical sides : Python, Pytorch, Git (could be learned on the fly), linear algebra (SVD...), differential calculus...
And many others more skills can be used and learned during the internship (see footnotes for some of the MVA related courses):

- Optimisation techniques [1]
- Properties of small or big networks [2]
- Expressivity of neural networks and general notions of expressivity (VC-dimension, Rademacher complexity) [3]

[1] Convex optimisation, Computational Statistics
[2] Deep learning in practice
[3] Fondements Théoriques du deep learning, Introduction to statistical learning

- Estimation of carbon footprint of a compuatation [4]
- Deep understanding of computationnal cost of neural network training and linear algebra operation on GPU [5]
- Many softwear developement skills (continuous integration, documention, use of cluster (slurm), ...) to improve our open-source implementation [6]

References

[GJS+20] Mario Geiger, Arthur Jacot, Stefano Spigler, Franck Gabriel, Levent Sagun, Stéphane d'Ascoli, Giulio Biroli, Clément Hongler, and Matthieu Wyart. Scaling description of generalization with number of parameters in deep learning. *Journal of Statistical Mechanics: Theory and Experiment*, 2020(2):023401, February 2020.

[LSY19] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable Architecture Search, April 2019.

[LUW17] Christos Louizos, Karen Ullrich, and Max Welling. Bayesian Compression for Deep Learning, November 2017.

[MRLW22] Kaitlin Maile, Emmanuel Rachelson, Hervé Luga, and Dennis G. Wilson. When, where, and how to add new neurons to ANNs, May 2022.

[VRCC24] Manon Verbockhaven, Théo Rudkiewicz, Sylvain Chevallier, and Guillaume Charpiat. Growing tiny networks: Spotting expressivity bottlenecks and fixing them optimally. *Transactions on Machine Learning Research*, 2024.

[4] Intelligence Artificielle et Environnement

[5] Geometric data analysis

[6] Reproducible research project, Fundamentals of reproducible research and free software