Growing Tiny Networks: Spotting Expressivity Bottlenecks and Fixing Them Optimally

Anonymous authors
Paper under double-blind review

Abstract

Machine learning tasks are generally formulated as optimization problems, where one searches for an optimal function within a certain functional space. In practice, parameterized functional spaces are considered, in order to be able to perform gradient descent. Typically, a neural network architecture is chosen and fixed, and its parameters (connection weights) are optimized, yielding an architecture-dependent result. This way of proceeding however forces the evolution of the function during training to lie within the realm of what is expressible with the chosen architecture, and prevents any optimization across architectures. Costly architectural hyper-parameter optimization is often performed to compensate for this. Instead, we propose to adapt the architecture on the fly during training. We show that the information about desirable architectural changes, due to expressivity bottlenecks when attempting to follow the functional gradient, can be extracted from backpropagation. To do this, we propose a mathematical definition of expressivity bottlenecks, which enables us to detect, quantify and solve them while training, by adding suitable neurons. Thus, while the standard approach requires large networks, in terms of number of neurons per layer, for expressivity and optimization reasons, we provide tools and properties to develop an architecture starting with a very small number of neurons. As a proof of concept, we show results on the CIFAR dataset, matching large neural network accuracy, with competitive training time, while removing the need for standard architectural hyper-parameter search.

1 Introduction

Issues with the fixed-architecture paradigm. Universal approximation theorems such as (Hornik et al., 1989; Cybenko, 1989) are historically among the first theoretical results obtained on neural networks, stating the family of neural networks with arbitrary width as a good candidate for a parameterized space of functions to be used in machine learning. However the current common practice in neural network training consists in choosing a fixed architecture, and training it, without any possible architecture modification meanwhile. This inconveniently prevents the direct application of these universal approximation theorems, as expressivity bottlenecks that might arise in a given layer during training will not be able to be fixed. There are two approaches to circumvent this in daily practice. Either one chooses a (very) large width, to be sure to avoid expressivity and optimization issues (Hanin & Rolnick, 2019b; Raghu et al., 2017), to the cost of extra computational power consumption for training and applying such big models; to mitigate this cost, model reduction techniques are often used afterwards, using pruning, tensor factorization, quantization (Louizos et al., 2017) or distillation (Hinton et al., 2015). Or one tries different architectures and keeps the most suitable one (in terms of performance-size compromise for instance), which multiplies the computational burden by the number of trials. This latter approach relates to the Auto-DeepLearning field (Liu et al., 2020), where different exploration strategies over the space of architecture hyper-parameters (among other ones) have been tested, including reinforcement learning (Baker et al., 2017; Zoph & Le, 2016), Bayesian optimization techniques (Mendoza et al., 2016), and evolutionary approaches (Miller et al., 1989; Stanley et al., 2009; Miikkulainen et al., 2017; Bennet et al., 2021), that all rely on random tries and consequently take time for exploration. Within that line, Net2Net (Chen et al., 2015), AdaptNet (Yang et al., 2018) and MorphNet (Gordon et al., 2018) propose different strategies to explore possible variations of a given

architecture, possibly guided by model size constraints. Instead, we aim at providing a way to locate precisely expressivity bottlenecks in a trained network, which might speed up neural architecture search significantly. Moreover, based on such observations, we aim at modifying the architecture *on the fly* during training, in a single run (no re-training), using first-order derivatives only, while avoiding neuron redundancy. Related work on architecture adaptation while training includes probabilistic edges (Liu et al., 2019) or sparsifying priors (Wolinski et al., 2020). Yet the training is done on the largest architecture allowed, which is resource-consuming. On the opposite we aim at starting from the simplest architecture possible.

Optimization properties. An important reason for common practice to choose wide architectures is the associated optimization properties: sufficiently larger networks are proved theoretically and shown empirically to be better optimized than small ones (Jacot et al., 2018). Typically, small networks exhibit issues with spurious local minima, while wide ones find good nearly-global minima. One of our goals is to train small networks without suffering from such optimization difficulties.

Neural architecture growth. A related line of work consists in growing networks neuron by neuron, by iteratively estimating the best possible neurons to add, according to a certain criterion. For instance, approaches such as (Wu et al., 2019) or Firefly (Wu et al., 2020) aim at escaping local minima by adding neurons that minimize the loss under neighborhood constraints. These neurons are found by gradient descent or by solving quadratic problems involving second-order derivatives. Other approaches (Causse et al., 2019; Bashtova et al., 2022), including GradMax (Evci et al., 2022), seek to minimize the loss as fast as possible and involve another quadratic problem. However the neurons added by these approaches are possibly redundant with existing neurons, especially if one does not wait for training convergence to a local minimum (which is time consuming) before adding neurons, therefore producing larger-than-needed architectures.

Redundancy. To our knowledge, the only approach tackling redundancy in neural architecture growth adds random neurons that are orthogonal in some sense to the ones already present (Maile et al., 2022). More precisely, the new neurons are picked within the *kernel* (preimage of $\{0\}$) of an application describing already existing neurons. Two such applications are proposed, respectively the matrix of fan-in weights and the pre-activation matrix, yielding two different notions of orthogonality. The latter formulation is close to the one of GradMax, in that both study first-order loss variations and use the same pre-activation matrix, with an important difference though: GradMax optimally decreases the loss without caring about redundancy, while the other one avoids redundancy but picks random directions instead of optimal ones. In this paper we bridge the gap between these two approaches, picking optimal directions that avoid redundancy in the pre-activation space.

Notions of expressivity. Several concepts of expressivity or complexity exist in the Machine Learning literature, ranging from Vapnik-Chervonenkis dimension (Vapnik & Chervonenkis, 1971) and Rademacher complexity (Koltchinskii, 2001) to the number of pieces in a piecewise affine function (as networks with ReLU activations are) (Serra et al., 2018; Hanin & Rolnick, 2019a). Bottlenecks have been also studied from the point of view of Information Theory, through mutual information between the activities of different layers (Tishby & Zaslavsky, 2015; Dai et al., 2018); this quantity is difficult to estimate though. Also relevant and from Information Theory, the Minimum Description Length paradigm and Kolmogorov complexity (Kolmogorov, 1965; Li et al., 2008) enable to define trade-offs between performance and model complexity.

In this article, we aim at measuring lacks of expressivity as the difference between what the backpropagation asks for and what can be done by a small parameter update (such as a gradient step), that is, between the desired variation for each activation in each layer (for each sample) and the best one that can be realized by a parameter update. Intuitively, differences arise when a layer does not have sufficient expressive power to realize the desired variation. Our main contributions are that we:

- adopt a functional analysis perspective on gradient descent in neural networks, advocating to follow the functional gradient. We not only optimize the weights of the current architecture but also dynamically adjust the architecture itself to progress towards a suitable parameterized functional spaces. This approach mitigates optimization challenges like local minima that are due to thin architectures;

- properly define and quantify the concept of expressivity bottlenecks, both globally at the neural network output and locally at individual layers, in a computationally accessible manner. This methodology enables to localize expressivity bottlenecks within a neural network;
- formally define as a quadratic problem the best possible neurons to add to a given layer to decrease lacks of expressivity ; solve it and compute the associated expressivity gain;
- use a naive strategy on the basis of our approach to expand a neural network while maintaining competitive computational complexity and performance compared to large and well-known model or other NAS search methods.

2 Main concepts

2.1 Notations

Let $\mathcal{F}$ be a functional space, e.g. $L_2(\mathbb{R}^p \to \mathbb{R}^d)$, and a loss function $\mathcal{L} : \mathcal{F} \to \mathbb{R}^+$ defined on it, of the form $\mathcal{L}(f) = \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim\mathcal{D}}\left[\ell(f(\boldsymbol{x}), \boldsymbol{y})\right]$, where ℓ is the per-sample loss, assumed to be differentiable, and where $\mathcal{D}$ is the sample distribution, from which the dataset $\{(\boldsymbol{x}_1, \boldsymbol{y}_1), ..., (\boldsymbol{x}_N, \boldsymbol{y}_N)\}$ is sampled, with $\boldsymbol{x}_i \in \mathbb{R}^p$ and $\boldsymbol{y}_i \in \mathbb{R}^d$.

For the sake of simplicity we consider a feedforward neural network $f_\theta : \mathbb{R}^p \to \mathbb{R}^d$ with L hidden layers, each of which consisting of an affine layer with weights $\boldsymbol{W}_l$ followed by a differentiable activation function σ_l which satisfies $\sigma_l(0) = 0$. The network parameters are then $\theta := (\boldsymbol{W}_l)_{l=1...L}$. The network iteratively computes:

$$\boldsymbol{b}_0(\boldsymbol{x}) = \begin{pmatrix} \boldsymbol{x} \\ 1 \end{pmatrix}$$

$$\forall l \in [1, L], \quad \begin{cases} \boldsymbol{a}_l(\boldsymbol{x}) &= \boldsymbol{W}_l\, \boldsymbol{b}_{l-1}(\boldsymbol{x}) \\ \boldsymbol{b}_l(\boldsymbol{x}) &= \begin{pmatrix} \sigma_l(\boldsymbol{a}_l(\boldsymbol{x})) \\ 1 \end{pmatrix} \end{cases}$$

$$f_\theta(\boldsymbol{x}) = \sigma_L(\boldsymbol{a}_L(\boldsymbol{x}))$$

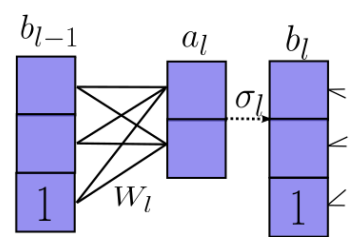

Figure 1: Notations

To any vector-valued function noted $\boldsymbol{t}(\boldsymbol{x})$ and any batch of inputs $\boldsymbol{X} := [\boldsymbol{x}_1, ..., \boldsymbol{x}_n]$, we associate the concatenated matrix $\boldsymbol{T}(\boldsymbol{X}) := \begin{pmatrix} \boldsymbol{t}(\boldsymbol{x}_1) & ... & \boldsymbol{t}(\boldsymbol{x}_n) \end{pmatrix} \in \mathbb{R}^{|\boldsymbol{t}(.)| \times n}$. The matrices of pre-activation and post-activation activities at layer l over a minibatch $\boldsymbol{X}$ are thus respectively: $\boldsymbol{A}_l(\boldsymbol{X}) = \begin{pmatrix} \boldsymbol{a}_l(\boldsymbol{x}_1) & ... & \boldsymbol{a}_l(\boldsymbol{x}_n) \end{pmatrix}$ and $\boldsymbol{B}_l(\boldsymbol{X}) = \begin{pmatrix} \boldsymbol{b}_l(\boldsymbol{x}_1) & ... & \boldsymbol{b}_l(\boldsymbol{x}_n) \end{pmatrix}$.

NB: convolutions can also be considered, with appropriate representations (cf matrix $\boldsymbol{b}_l^c(\boldsymbol{x})$ in 46).

2.2 Approach

Functional gradient descent. We take a functional perspective on the use of neural networks. Ideally in a machine learning task, one would search for a function $f : \mathbb{R}^p \to \mathbb{R}^d$ that minimizes the loss $\mathcal{L}$ by gradient descent: $\frac{\partial f}{\partial t} = -\nabla_f \mathcal{L}(f)$ for some metric on the functional space $\mathcal{F}$ (typically, $L_2(\mathbb{R}^p \to \mathbb{R}^d)$), where ∇_f denotes the functional gradient and t denotes the evolution time of the gradient descent. The descent direction $\boldsymbol{v}_{\text{goal}} := -\nabla_f \mathcal{L}(f)$ is a function of the same type as f and whose value at $\boldsymbol{x}$ is easily computable as $\boldsymbol{v}_{\text{goal}}(\boldsymbol{x}) = -\left(\nabla_f \mathcal{L}(f)\right)(\boldsymbol{x}) = -\nabla_{\boldsymbol{u}} \ell(\boldsymbol{u}, \boldsymbol{y}(\boldsymbol{x}))\big|_{\boldsymbol{u}=f(\boldsymbol{x})}$ (see Appendix A.1 for more details). This direction $\boldsymbol{v}_{\text{goal}}$ is the best infinitesimal variation in $\mathcal{F}$ to add to f to decrease the loss $\mathcal{L}$.

Parametric gradient descent reminder. However in practice, to represent functions and to compute gradients, the infinite-dimensioned functional space $\mathcal{F}$ has to be replaced with a finite-dimensioned parametric space of functions, which is usually done by choosing a particular neural network architecture $\mathcal{A}$ with weights $\theta \in \Theta_{\mathcal{A}}$. The associated parametric search space $\mathcal{F}_{\mathcal{A}}$ then consists of all possible functions f_θ that can be

represented with such a network for any parameter value θ. Under standard weak assumptions (see Appendix A.2), the gradient descent is of the form:

$$\frac{\partial \theta}{\partial t} = -\nabla_\theta \mathcal{L}(f_\theta) = -\mathop{\mathbb{E}}_{(\boldsymbol{x},\boldsymbol{y})\sim\mathcal{D}}\left[\nabla_\theta \ell(f_\theta(\boldsymbol{x}), \boldsymbol{y})\right]. \tag{1}$$

Using the chain rule (on $\frac{\partial f_\theta}{\partial t}$ then on $\nabla_\theta \ell(f_\theta(\boldsymbol{x}), \boldsymbol{y})$), these parameter updates yield a functional evolution:

$$\boldsymbol{v}_{\text{GD}} := \frac{\partial f_\theta}{\partial t} = \frac{\partial f_\theta}{\partial \theta}\frac{\partial \theta}{\partial t} = \frac{\partial f_\theta}{\partial \theta} \mathop{\mathbb{E}}_{(\boldsymbol{x},\boldsymbol{y})\sim\mathcal{D}}\left[\frac{\partial f_\theta}{\partial \theta}^T(\boldsymbol{x})\, \boldsymbol{v}_{\text{goal}}(\boldsymbol{x})\right] \tag{2}$$

which significantly differs from the original functional gradient descent. We will aim to augment the neural network architecture so that parametric gradient descents can get closer to the functional one.

Optimal move direction. We name $\mathcal{T}_{\mathcal{A}}^{f_\theta}$, or just $\mathcal{T}_{\mathcal{A}}$, the tangent space of $\mathcal{F}_{\mathcal{A}}$ at f_θ, that is, the set of all possible infinitesimal variations around f_θ under small parameter variations:

$$\mathcal{T}_{\mathcal{A}}^{f_\theta} := \left\{ \frac{\partial f_\theta}{\partial \theta}\,\delta\theta \;\middle|\; \text{s.t. } \delta\theta \in \Theta_{\mathcal{A}} \right\}$$

This linear [1]functional space is a first-order approximation of the neighborhood of f_θ within $\mathcal{F}_{\mathcal{A}}$. The direction $\boldsymbol{v}_{\text{GD}}$ obtained above by gradient descent is actually not the best one to consider within $\mathcal{T}_{\mathcal{A}}$. Indeed, the best move $\boldsymbol{v}^*$ would be the orthogonal projection of the desired direction $\boldsymbol{v}_{\text{goal}} := -\nabla_{f_\theta}\mathcal{L}(f_\theta)$ onto $\mathcal{T}_{\mathcal{A}}$. This projection is what a (generalization of the notion of) natural gradient would compute (Ollivier, 2017).

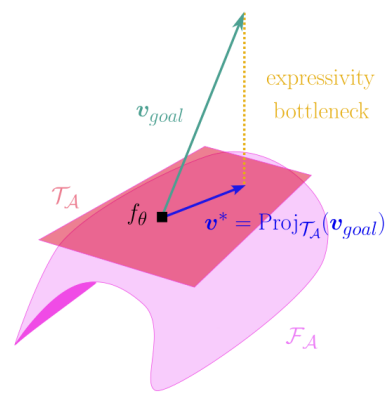

Figure 2: Expressivity bottleneck

Indeed, the parameter variation $\delta\theta^*$ associated to the functional variation $\boldsymbol{v}^* = \frac{\partial f_\theta}{\partial \theta}\delta\theta^*$ is the gradient $-\nabla_\theta^{\mathcal{T}_{\mathcal{A}}}\mathcal{L}(f_\theta)$ of $\mathcal{L}\circ f_\theta$ w.r.t. parameters θ when considering the L_2 metric on *functional* variations $\|\frac{\partial f_\theta}{\partial \theta}\delta\theta\|_{L_2(\mathcal{T}_{\mathcal{A}})}$, not to be confused with the usual gradient $\nabla_\theta\mathcal{L}(f_\theta)$, based on the L_2 metric on *parameter* variations $\|\delta\theta\|_{L_2(\mathbb{R}^{|\Theta_{\mathcal{A}}|})}$. This can be seen in a proximal formulation as:

$$\boldsymbol{v}^* = \mathop{\arg\min}_{\boldsymbol{v}\in\mathcal{T}_{\mathcal{A}}} \|\boldsymbol{v} - \boldsymbol{v}_{\text{goal}}\|^2 = \mathop{\arg\min}_{\boldsymbol{v}\in\mathcal{T}_{\mathcal{A}}}\left\{ D_f\mathcal{L}(f)(\boldsymbol{v}) + \frac{1}{2}\|\boldsymbol{v}\|^2 \right\} \tag{3}$$

where D is the directional derivative (see details in Appendix A.3), or equivalently as:

$$\delta\theta^* = \mathop{\arg\min}_{\delta\theta\in\Theta_{\mathcal{A}}}\left\|\frac{\partial f_\theta}{\partial \theta}\delta\theta - \boldsymbol{v}_{\text{goal}}\right\|^2 = \mathop{\arg\min}_{\delta\theta\in\Theta_{\mathcal{A}}}\left\{ D_\theta\mathcal{L}(f_\theta)(\delta\theta) + \frac{1}{2}\left\|\frac{\partial f_\theta}{\partial \theta}\delta\theta\right\|^2 \right\} =: -\nabla_\theta^{\mathcal{T}_{\mathcal{A}}}\mathcal{L}(f_\theta)\,.$$

Lack of expressivity. When $\boldsymbol{v}_{\text{goal}}$ does not belong to the reachable subspace $\mathcal{T}_{\mathcal{A}}$, there is a lack of expressivity, that is, the parametric space $\mathcal{A}$ is not rich enough to follow the ideal functional gradient descent. This happens frequently with small neural networks (see Appendix A.4 for an example). The expressivity bottleneck is then quantified as the distance $\|\boldsymbol{v}^* - \boldsymbol{v}_{\text{goal}}\|$ between the functional gradient $\boldsymbol{v}_{\text{goal}}$ and the optimal functional move $\boldsymbol{v}^*$ given the architecture $\mathcal{A}$ (in the sense of Eq. 3).

[1] $\frac{\partial f_\theta}{\partial \theta}\,\delta\theta : \mathbb{R}^p \to \mathbb{R}^d \;\; \boldsymbol{x} \mapsto \frac{\partial f_\theta(\boldsymbol{x})}{\partial \theta}\,\delta\theta$

2.3 Generalizing to all layers

Ideal updates. The same reasoning can be applied to the pre-activations $\boldsymbol{a}_l$ at each layer l, seen as functions $\boldsymbol{a}_l : \boldsymbol{x} \in \mathbb{R}^p \mapsto \boldsymbol{a}_l(\boldsymbol{x}) \in \mathbb{R}^{d_l}$ defined over the input space of the neural network. The optimal parameter update for a given layer l then follows the projection of the desired update $-\nabla_{\boldsymbol{a}_l}\mathcal{L}(f_\theta)$ of the pre-activation functions $\boldsymbol{a}_l$ onto the linear subspace $\mathcal{T}_{\mathcal{A}}^{\boldsymbol{a}_l}$ of pre-activation variations that are possible with the architecture, as we will detail now.

Given a sample $(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{D}$, standard backpropagation already iteratively computes $\boldsymbol{v}^l_{\text{goal}}(\boldsymbol{x}) := -\left(\nabla_{\boldsymbol{a}_l}\mathcal{L}(f_\theta)\right)(\boldsymbol{x}) = -\nabla_{\boldsymbol{u}}\ell\left(\sigma_L(\boldsymbol{W}_L\,\sigma_{L-1}(\boldsymbol{W}_{L-1}\dots\sigma_l(\boldsymbol{u}))), \boldsymbol{y}\right)|_{\boldsymbol{u}=\boldsymbol{a}_l(\boldsymbol{x})}$, which is the derivative of the loss $\ell(f_\theta(\boldsymbol{x}), \boldsymbol{y})$ with respect to the pre-activations $\boldsymbol{u} = \boldsymbol{a}_l(\boldsymbol{x})$ of each layer. This is usually performed in order to compute the gradients w.r.t. model parameters $\boldsymbol{W}_l$, as $\nabla_{\boldsymbol{W}_l}\ell(f_\theta(\boldsymbol{x}), \boldsymbol{y}) = \frac{\partial \boldsymbol{a}_l(\boldsymbol{x})}{\partial W_l}\,\nabla_{\boldsymbol{a}_l}\ell(f_\theta(\boldsymbol{x}), \boldsymbol{y})$.

$\boldsymbol{v}^l_{\text{goal}}(\boldsymbol{x}) := -\left(\nabla_{\boldsymbol{a}_l}\mathcal{L}(f_\theta)\right)(\boldsymbol{x})$ indicates the direction in which one would like to change the layer pre-activations $\boldsymbol{a}_l(\boldsymbol{x})$ in order to decrease the loss at point $\boldsymbol{x}$. However, given a minibatch of points $(\boldsymbol{x}_i)$, most of the time no parameter move $\delta\theta$ is able to induce this progression for each $\boldsymbol{x}_i$ simultaneously, because the θ-parameterized family of functions $\boldsymbol{a}_l$ is not expressive enough.

Activity update resulting from a parameter change. Given a subset of parameters $\tilde{\theta}$ (such as the ones specific to a layer: $\tilde{\theta} = \boldsymbol{W}_l$), and an incremental direction $\delta\tilde{\theta}$ to update these parameters (e.g. the one resulting from a gradient descent: $\delta\tilde{\theta} = -\sum_{(\boldsymbol{x},\boldsymbol{y})\in\text{minibatch}}\nabla_{\tilde{\theta}}\ell(f_\theta(\boldsymbol{x}), \boldsymbol{y})$, the impact of the parameter update $\gamma\delta\tilde{\theta}$ on the pre-activations $\boldsymbol{a}_l$ at layer l at order 1 in γ, where γ an amplitude factor, is as $\boldsymbol{v}^l(\boldsymbol{x}, \gamma\delta\tilde{\theta}) := \gamma\frac{\partial \boldsymbol{a}_l(\boldsymbol{x})}{\partial\tilde{\theta}}\,\delta\tilde{\theta}$. The factor γ is the analogue of the learning rate and is consider to be small.

3 Expressivity bottlenecks

We now quantify expressivity bottlenecks at any layer l as the distance between the desired activity update $\boldsymbol{v}^l_{\text{goal}}(.)$ and the best realizable one $\boldsymbol{v}^l(.)$ (cf Figure 2):

Definition 3.1 (Lack of expressivity)**.** *For a neural network f_θ and a minibatch of points $\boldsymbol{X} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^n$, we define the lack of expressivity at layer l as how far the desired activity update $\boldsymbol{V}^l_{goal} = (\boldsymbol{v}^l_{goal}(\boldsymbol{x}_1), \boldsymbol{v}^l_{goal}(\boldsymbol{x}_2), \dots)$ is from the closest possible activity update $V^l = (\boldsymbol{v}^l(\boldsymbol{x}_1), \boldsymbol{v}^l(\boldsymbol{x}_2), \dots)$ realizable by a parameter change $\delta\theta$:*

$$\Psi^l := \min_{\boldsymbol{v}^l\in\mathcal{T}_{\mathcal{A}}^{\boldsymbol{a}_l}} \frac{1}{n}\sum_{i=1}^n \left\|\boldsymbol{v}^l(\boldsymbol{x}_i) - \boldsymbol{v}^l_{\text{goal}}(\boldsymbol{x}_i)\right\|^2 = \min_{\delta\theta}\frac{1}{n}\left\|\boldsymbol{V}^l(\boldsymbol{X}, \delta\theta) - \boldsymbol{V}^l_{\text{goal}}(\boldsymbol{X})\right\|^2_{\text{Tr}} \tag{4}$$

where $||.||$ stands for the L_2 norm, $||.||_{\text{Tr}}$ for the Frobenius norm, and $\boldsymbol{V}^l(\boldsymbol{X}, \delta\theta)$ is the activity update resulting from parameter change $\delta\theta$ as defined in previous section. In the two following parts we fix the minibatch $\boldsymbol{X}$ and simplify notations accordingly by removing the dependency on $\boldsymbol{X}$.

3.1 Best move without modifying the architecture of the network

Let $\delta\boldsymbol{W}_l^*$ be the solution of equation (4) when the parameter variation $\delta\theta$ is restricted to involve only layer l parameters, i.e. $\boldsymbol{W}_l$. This move is sub-optimal in that it does not result from an update of all architecture parameters but only of the current layer ones:

$$\delta\boldsymbol{W}_l^* = \arg\min_{\delta\boldsymbol{W}}\frac{1}{n}\left\|\boldsymbol{V}^l(\delta\boldsymbol{W}) - \boldsymbol{V}^l_{\text{goal}}\right\|^2_{\text{Tr}} \tag{5}$$

Proposition 3.1. *The solution of Problem (5) is: $\delta\boldsymbol{W}_l^* = \frac{1}{n}\boldsymbol{V}^l_{goal}\boldsymbol{B}_{l-1}^T(\frac{1}{n}\boldsymbol{B}_{l-1}\boldsymbol{B}_{l-1}^T)^+$ where P^+ denotes the generalized inverse of matrix P.*

This update $\delta\boldsymbol{W}_l^*$ is not equivalent to the usual gradient descent update, whose form is $\delta\boldsymbol{W}_l^{\text{GD}} \propto \boldsymbol{V}^l_{\text{goal}}\boldsymbol{B}_{l-1}^T$. In fact the associated activity variation, $\delta\boldsymbol{W}_l^*\boldsymbol{B}_{l-1}$, is the projection of $\boldsymbol{V}^l_{\text{goal}}$ on the post-activation matrix of layer $l-1$, that is to say onto the span of all possible post-activation directions, through the projector

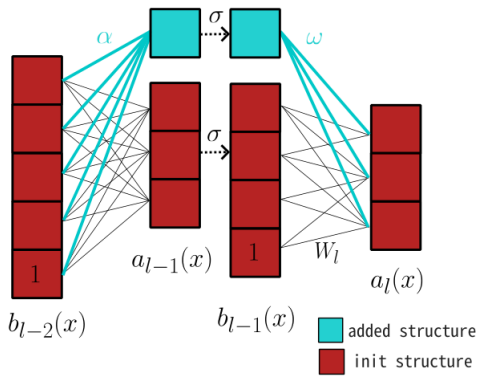

Figure 3: Adding one neuron to layer l in cyan ($K = 1$), with connections in cyan. Here, $\boldsymbol{\alpha} \in \mathrm{R}^5$ and $\boldsymbol{\omega} \in \mathrm{R}^3$.

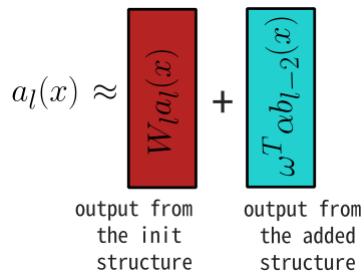

Figure 4: Sum of functional moves

$\frac{1}{n}\boldsymbol{B}_{l-1}^T(\frac{1}{n}\boldsymbol{B}_{l-1}\boldsymbol{B}_{l-1}^T)^+\boldsymbol{B}_{l-1}$. To increase expressivity if needed, we will aim at increasing this span with the most useful directions to close the gap between this best update and the desired one. Note that the update $\delta\boldsymbol{W}_l^*$ consists of a standard gradient ($\boldsymbol{V}_{\text{goal}}^l\boldsymbol{B}_{l-1}^T$) and of a (kind of) natural gradient only for the last part (projector), as we consider metrics in the pre-activation space.

3.2 Reducing expressivity bottleneck by modifying the architecture

To get as close as possible to $\boldsymbol{V}_{\text{goal}}^l$ and to increase the expressive power of the current neural network, we modify each layer of its structure. At layer $l-1$, we add K neurons $n_1, ..., n_K$ with input weights $\boldsymbol{\alpha}_1, ..., \boldsymbol{\alpha}_k$ and output weights $\boldsymbol{\omega}_1, ..., \boldsymbol{\omega}_K$ (cf Figure 3). We have the following expansions by concatenation: $\boldsymbol{W}_{l-1}^T \leftarrow \begin{pmatrix}\boldsymbol{W}_{l-1}^T & \boldsymbol{\alpha}_1 & ... & \boldsymbol{\alpha}_K\end{pmatrix}$ and $\boldsymbol{W}_l \leftarrow \begin{pmatrix}\boldsymbol{W}_l & \boldsymbol{\omega}_1 & ... & \boldsymbol{\omega}_K\end{pmatrix}$. We note this architecture modification $\theta \leftarrow \theta \oplus \theta_{\leftrightarrow}^K$ where $\oplus$ is the concatenation sign and $\theta_{\leftrightarrow}^K := (\boldsymbol{\alpha}_k, \boldsymbol{\omega}_k)_{k=1}^K$ are the K added neurons.

The added neurons could be chosen randomly, as in usual neural network initialization, but this would not yield any guarantee regarding the impact on the system loss. Another possibility would be to set either input weights $(\boldsymbol{\alpha}_k)_{k=1}^K$ or output weights $(\boldsymbol{\omega}_k)_{k=1}^K$ to 0, so that the function $f_\theta(.)$ would not be modified, while its gradient w.r.t. θ would be enriched from the new parameters. Another option is to solve a optimization problem as in the previous section with the modified structure $\theta \leftarrow \theta \oplus \theta_{\leftrightarrow}^K$ and jointly search for both the optimal new parameters $\theta_{\leftrightarrow}^K$ and the optimal variation $\delta\boldsymbol{W}$ of the old ones:

$$\underset{\theta_{\leftrightarrow}^K,\, \delta\boldsymbol{W}}{\arg\min} \frac{1}{n} \left\| \boldsymbol{V}^l(\delta\boldsymbol{W} \oplus \theta_{\leftrightarrow}^K) - \boldsymbol{V}_{\text{goal}}^l \right\|_{\text{Tr}}^2 \tag{6}$$

As shown in figure 4, the displacement $\boldsymbol{V}^l$ at layer l is actually a sum of the moves induced by the neurons already present ($\delta\boldsymbol{W}$) and by the added neurons ($\theta_{\leftrightarrow}^K$), our problem rewrites as :

$$\underset{\theta_{\leftrightarrow}^K,\, \delta\boldsymbol{W}}{\arg\min} \frac{1}{n} \left\| \boldsymbol{V}^l(\theta_{\leftrightarrow}^K) + \boldsymbol{V}^l(\delta\boldsymbol{W}) - \boldsymbol{V}_{\text{goal}}{}^l \right\|_{\text{Tr}}^2 \tag{7}$$

with $\boldsymbol{v}^l(\boldsymbol{x}, \theta_{\leftrightarrow}^K) := \sum_{k=1}^K \boldsymbol{\omega}_k \left(b_{l-2}(\boldsymbol{x})^T\boldsymbol{\alpha}_k\right)$ (See A.5). We choose $\delta\boldsymbol{W}$ as the best move of already-existing parameters as defined in Proposition 3.1 and we note $\boldsymbol{V}_{\text{goal}_{proj}}^l := \boldsymbol{V}_{\text{goal}}^l - \boldsymbol{V}^l(\delta\boldsymbol{W}^*)$. We are looking for the solution $\left(K^*, \theta_{\leftrightarrow}^{K*}\right)$ of the optimization problem :

$$\underset{K,\, \theta_{\leftrightarrow}^K}{\arg\min} \frac{1}{n} \left\| \boldsymbol{V}^l(\theta_{\leftrightarrow}^K) - \boldsymbol{V}_{\text{goal}_{proj}}^l \right\|_{\text{Tr}}^2 . \tag{8}$$

One should note that problems (7) and (8) are generally not equivalent, though similar (cf C.4).This quadratic optimization problem can be solved thanks to the low-rank matrix approximation theorem (Eckart & Young,

1936), using matrices $\boldsymbol{N} := \frac{1}{n}\boldsymbol{B}_{l-2}\left(\boldsymbol{V}^l_{\text{goal}_{proj}}\right)^T$ and $\boldsymbol{S} := \frac{1}{n}\boldsymbol{B}_{l-2}\boldsymbol{B}_{l-2}^T$. As $\boldsymbol{S}$ is semi-positive definite, let its truncated SVD be $\boldsymbol{S} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{U}^T$, and define $\boldsymbol{S}^{-\frac{1}{2}} := \boldsymbol{U}\sqrt{\boldsymbol{\Sigma}}^{-1}\boldsymbol{U}^T$, with the convention that the inverse of 0 eigenvalues is 0. Finally, consider the truncated SVD of matrix $\boldsymbol{S}^{-\frac{1}{2}}\boldsymbol{N} = \sum_{k=1}^{R}\lambda_k \boldsymbol{u}_k \boldsymbol{v}_k^T$, where R is the rank of the matrix $\boldsymbol{S}^{-\frac{1}{2}}\boldsymbol{N}$. Then:

Proposition 3.2. *The solution of Problem (8) is:*

- *optimal number of neurons:* $K^* = R$
- *their optimal weights:* $\theta_{\leftrightarrow}^{K^*} = (\boldsymbol{\alpha}_k^*, \boldsymbol{\omega}_k^*)_k^{K^*} = \left(\sqrt{\lambda_k}\boldsymbol{S}^{-\frac{1}{2}}\boldsymbol{u}_k,\ \sqrt{\lambda_k}\boldsymbol{v}_k\right)_k^{K^*}$

Moreover for any number of neurons $K \leqslant R$, and associated weights $\theta_{\leftrightarrow}^{K,*}$, the expressivity gain can be quantified very simply as a function of the singular values λ_k:

$$\Psi^l_{\theta\oplus\theta_{\leftrightarrow}^{K,*}} = \Psi^l_\theta - \sum_{k=1}^{K}\lambda_k^2 \tag{9}$$

With an inequality ($\leq$) for convolutional layer.

In practice before adding new neurons (α, ω), we multiply them by an amplitude factor γ found by a simple line search (see Appendix E.3), i.e. we add $(\sqrt{\gamma}\alpha, \sqrt{\gamma}\omega)$. The addition of each neuron k has an impact on the bottleneck of the order of $\gamma\lambda_k^2$ provided γ is small. We observe the same phenomenon with the loss as stated in the next proposition :

Proposition 3.3. *Solving 8 is equivalent to minimizing the loss $\mathcal{L}$ at order one in $\boldsymbol{V}^l$. Furthermore performing an update of architecture with $\gamma\delta\boldsymbol{W}^*$ (5) and a neuron addition with $\gamma\theta_{\leftrightarrow}^{K^*}$ (3.2), has an impact on the loss at first order in γ as :*

$$\mathcal{L}(f_{\theta\oplus\theta_{\leftrightarrow}^K}) := \frac{1}{n}\mathcal{L}(f_{\theta\oplus\theta_{\leftrightarrow}^K}(\boldsymbol{x}_i), \boldsymbol{y}_i) = \mathcal{L}(f_\theta) - \gamma\left(\sigma'_{l-1}(0)\Delta_{\theta_{\leftrightarrow}^{K,*}} + \Delta_{\delta\boldsymbol{W}^*}\right) + o(\gamma) \tag{10}$$

With

$$\Delta_{\theta_{\leftrightarrow}^{K,*}} := \frac{1}{n}\left\langle \boldsymbol{V}^l_{goal_{proj}},\ \boldsymbol{V}^l(\theta_{\leftrightarrow}^{K,*})\right\rangle_{\text{Tr}} = \sum_{k=1}^{K}\lambda_k^2 \tag{11}$$

$$\Delta_{\delta\boldsymbol{W}^*} := \frac{1}{n}\left\langle \boldsymbol{V}^l_{goal},\ \boldsymbol{V}^l(\delta\boldsymbol{W}^*)\right\rangle_{\text{Tr}} \geq 0 \tag{12}$$

The λ_k could be used in a selection criterion realizing a trade-off with computational complexity. A selection based on statistical significance of singular values can also be performed. The full algorithm and its complexity are detailed in Appendices E.4 and E.5. We finish this section by some additional propositions and remarks.

Proposition 3.4. *If $\boldsymbol{S}$ is positive definite, then solving 8 is equivalent to taking $\boldsymbol{\omega}_k = \boldsymbol{N}\boldsymbol{\alpha}_k$ and finding the K first eigenvectors $\boldsymbol{\alpha}_k$ associated to the K largest eigenvalues λ of the generalized eigenvalue problem :*

$$\boldsymbol{N}\boldsymbol{N}^T\boldsymbol{\alpha}_k = \lambda\boldsymbol{S}\boldsymbol{\alpha}_k$$

Corollary 1. *For all integers m, m' such that $m + m' \leqslant R$, at order one in $\boldsymbol{V}$, adding $m + m'$ neurons simultaneously according to the previous method is equivalent to adding m neurons then m' neurons by applying successively the previous method twice.*

One should also note that the family $\{\boldsymbol{V}^{l+1}((\boldsymbol{\alpha}_k, \boldsymbol{\omega}_k))\}_{k=1}^K$ of pre-activity variations induced by adding the neurons $\theta_{\leftrightarrow}^{K,*}$ is orthogonal for the trace scalar product. We could say that the added neurons are orthogonal to each other (and to the already-present ones) in that sense. Interestingly, the GradMax method (Evci et al., 2022) also aims at minimizing the loss 132, but without avoiding redundancy (see Appendix B.1 for more details).

4 About greedy growth sufficiency and TINY convergence

One might wonder whether a greedy approach on layer growth might get stuck in a non-optimal state. By *greedy* we mean that every neuron added has to decrease the loss. Since in this work we add neurons layer per layer independently, we study here the case of a single hidden layer network, to spot potential layer growth issues. For the sake of simplicity, we consider the task of least square regression towards an explicit continuous target f^*, defined on a compact set. That is, we aim at minimizing the loss:

$$\inf_f \sum_{\boldsymbol{x} \in \mathcal{D}} \|f(\boldsymbol{x}) - f^*(\boldsymbol{x})\|^2 \tag{13}$$

where $f(\boldsymbol{x})$ is the output of the neural network and $\mathcal{D}$ is the training set. Proofs and supplementary propositions are deferred to Appendix D, in particular D.4 and D.7.

First, if one allows only adding neurons but no modification of already existing ones:

Proposition 4.1 (Exponential convergence to 0 training error by ReLU neuron additions)**.** *It is possible to decrease the loss exponentially fast with the number t of added neurons, i.e. as $\gamma^t \mathcal{L}(f)$, towards 0 training loss, and this in a greedy way, that is, such that each added neuron decreases the loss. The factor γ is $\gamma = 1 - \frac{1}{n^3 d'} \left(\frac{d_m}{d_M}\right)^2$, where d_m and d_M are quantities solely dependent on the dataset geometry, d' is the output dimension of the network, and n is the dataset size.*

In particular, there exists no situation where one would need to add many neurons simultaneously to decrease the loss: it is always feasible with a single neuron.

TINY might get stuck when no correlation between inputs $\boldsymbol{x}_i$ and desired output variations $f^*(\boldsymbol{x}_i) - f(\boldsymbol{x}_i)$ can be found anymore. To prevent this, one can choose an auxiliary method to add neurons in such cases, for instance random neurons (with a line search over their amplitude, cf. Appendix D.3), or locally-optimal neurons found by gradient descent, or solutions of higher-order expressivity bottleneck formulations using further developments of the activation function. We will name *completed-TINY* the completion of TINY by any such auxiliary method.

Now, if we also update already existing weights when adding new neurons, we get a stronger result:

Proposition 4.2 (Completed-TINY reaches 0 training error in at most n neuron additions)**.** *Under certain assumptions (full batch optimization, updating already existing parameters, and, more technically: polynomial activation function of order $\geqslant n^2$), completed-TINY reaches 0 training error in at most n neuron additions almost surely.*

Hence we see the importance of updating existing parameters on the convergence speed. This optimization protocol is actually the one we follow in practice when training neural networks with TINY (except when comparing with other methods using their protocol).

Note that our approach shares similarity with gradient boosting Friedman (2001) somehow, as we grow the architecture based on the gradient of the loss. Note also that finding the optimal neuron to add is actually NP-hard (Bach, 2017), but that we do not need new neuron optimality to converge to 0 training error.

5 Results

5.1 Comparison with GradMax on CIFAR-100

The closest growing method to TINY is GradMax (Evci et al. (2022)), as it solves a quadratic problem similar to (8). By construction, the objective of GradMax is to decrease the loss as fast as possible considering an infinitesimal increment of new neurons. The main difference is that GradMax does not take into account the expressivity of the current architecture as TINY does in (8) by projecting v_{goal}. In-depth details about the difference between the GradMax and TINY are provided in Appendix B.1.

In this section, we show on the CIFAR-100 dataset that solving (8) instead of B.1(defined by GradMax) to grow a network using a naive strategy allows better final performance and almost full expressivity power.

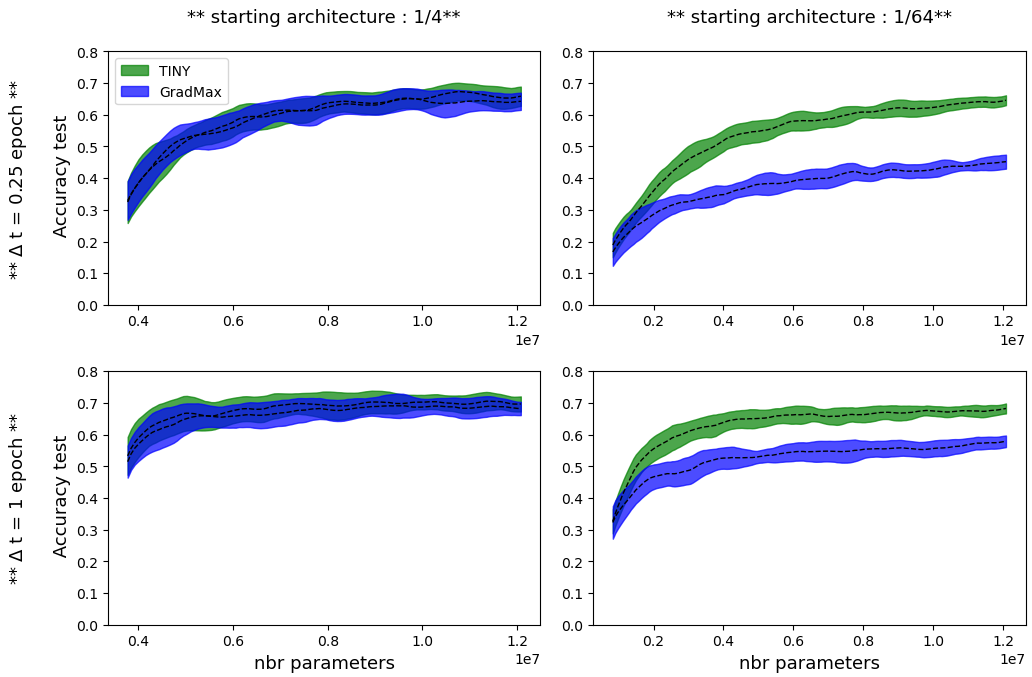

Figure 5: Test accuracy as a function of the number of parameters during architecture growth from ResNet_s to ResNet18. The left (resp. right) column is for the starting architecture $\text{ResNet}_{1/4}$(resp. $\text{ResNet}_{1/64}$). The upper (resp. lower) row is for Δt equal to 0.25 (resp. 1) epoch. Each line correspond to an independent run, 4 runs are performed for each settings.

To do so, we have re-implemented the GradMax method and mimicked its growing process which consists in increasing the architecture of a thin ResNet18 until it reaches the architecture of the usual ResNet18. This process is described in the pseudo code 1, where two parameters can be chosen : the relative thinness s of the starting architecture, w.r.t. the usual ResNet18 architecture 3 ($s = 1/4$ or $s = 1/64$), and the amount of training time between consecutive neuron additions ($\Delta t = 1$ or $\Delta t = 0.25$ epochs). Then the number of parameters and the performance of the growing network are evaluated at regular intervals to plot Figure 5. We note that both methods reach their final accuracy within less then one GPU days, outperforming by far others NAS search methods in their category (cf table 5).

Once the models have reached the final architecture ResNet18, they are trained for 250 epochs (or 500 epochs if they have not converged on the training set). We have summarized the final performance in Table 1. We also added the column Reference, which gives the performance of a ResNet18 trained from scratch by usual gradient descent with all its neurons. We do not expect TINY or GradMax to achieve the performance of the reference as its architecture and optimisation process have been optimised for years. The column small references corresponds to the training of the architecture ResNet18_s for $s = 1/4$ and $s = 1/64$ using standard gradient descent without any increase in architecture.

The details of the protocol can be found in the annexes F.1, as well as other technical details such as the dynamic of the training batch size E.2, the number of examples used to estimate and solve the expressivity bottleneck E.1 and the complexity of the algorithms E.5. For both methods, all the latter apply so that the main differences between GradMax and TINY in this experiment is the mathematical definition of the new neurons.

For $s = 1/64$, we observe a significant difference in performance between TINY and GradMax methods. While TINY models almost achieve the reference's performance, GradMax remain stuck 10 points below. This suggests that the framework proposed by GradMax is not sufficient to be able to start with an archi-

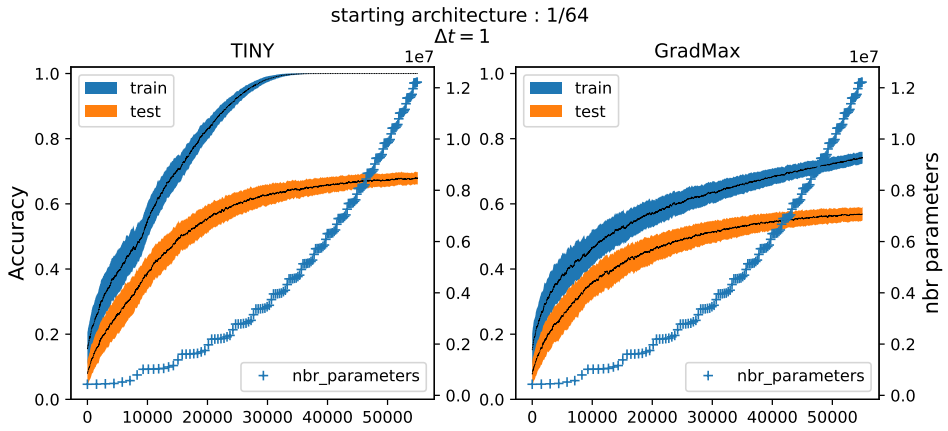

Figure 6: Evolution of accuracy and number of parameters as a function of gradient step for the setting $\Delta t = 1$, $s = 1/64$ for TINY and GradMax, mean and standard deviation over four runs. Other settings in the annexes 13

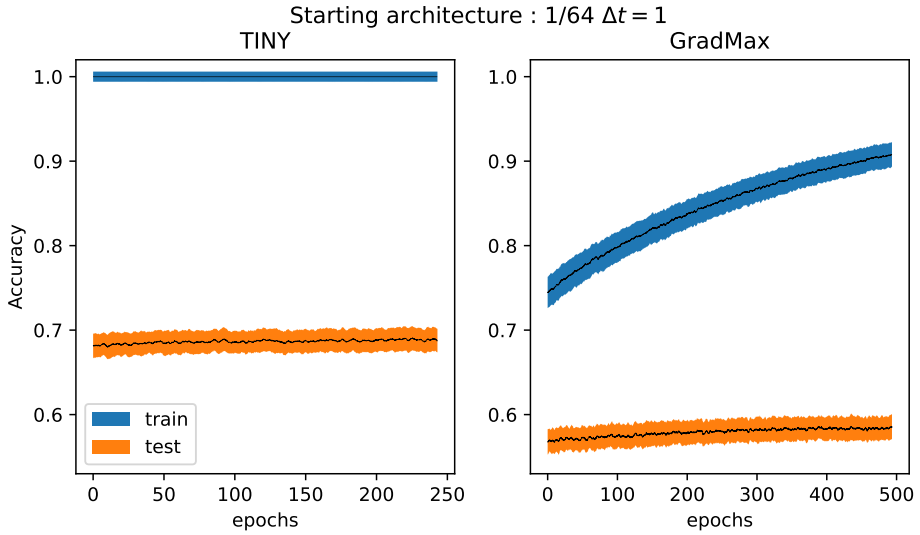

Figure 7: Evolution of accuracy and number of parameters as a function of gradient step for the setting $\Delta t = 1$, $s = 1/64$ during extra training for TINY and GradMax, mean and standard deviation over four runs. Other settings in the annexes 15 and 14.

		TINY		GradMax		Small references	Reference
	Δt	0.25	1	0.25	1		
s	1/4	67.2 ± 0.1	70.4 ± 0.2	65.1 ± 0.2	68.6 ± 0.1	68.6 ± 0.4 ^{5*}	72.8 ± 0.3 ^{5*}
	1/4	70.3 ± 0.2 ^{5*}	70.9 ± 0.2 ^{5*}	67.0 ± 0.2 ^{5*}	69.0 ± 0.2 ^{5*}		
	1/64	65.8 ± 0.1	68.1 ± 0.5	45.0 ± 0.4	56.8 ± 0.2	63.4 ± 0.3 ^{5*}	
	1/64	69.5 ± 0.2 ^{5*}	68.7 ± 0.6 ^{5*}	57.0 ± 0.4 ^{10*}	58.4 ± 0.2 ^{10*}		

Table 1: Final accuracy on test of ResNet18 after the architecture growth (*grey*) and after convergence (*black*). The number of stars indicates the multiple of 50 epochs needed to achieve convergence. With the starting architecture $\text{ResNet}_{1/64}$ and $\Delta t = 0.25$, the method TINY achieves 65.8 ± 0.1 on test after its growth and it reaches 69.5 ± 0.2 ^{5*}after $5* := 5 \times 50$ epochs (examples of training curves for the extra training in Figure 14). Mean and standard deviation are performed on 4 runs for each setting.

tecture far from full expressivity, i.e. $\text{ResNet}_{1/64}$, while TINY is able to handle it. As for the setting $s = 1/4$, both methods seem equivalent in terms of final performance and achieve full expressivity.

The curves on Figure 6, which are extracted from Figure 13 in the appendix, show that TINY models have converged at the end of the growing process, while GradMax ones have not. This latter effect contrasts with GradMax formulation which is to accelerate the gradient descent as fast as possible by adding neurons. Furthermore GradMax needs extra training to achieve full expressivity: for the particular setting $s = 1/64, \Delta t = 1$, the extra training time required by GradMax is twice as high as TINY's, as shown in Figure 7. This need for extra training also appears for all settings in Table 1. In particular for $s = 1/64, \Delta t = 0.25$, the difference in performance after and before extra training goes up to 20 % above the initial performance while it is only of 6% for TINY.

5.2 Comparison with Random on CIFAR-100 : initialisation impact

In this section, we focus on the impact of the new neurons' initialization. To do so, we consider as a baseline the Random method, which initializes the new neurons according to a Gaussian distribution: $(\alpha_k^*, \omega_k^*)_{k=1}^K \sim \mathcal{N}(0, I_d)$ or a uniform distribution $\mathcal{U}[-1, 1]$. Also, when adding new neurons, instead of normalizing them as previously, we search for the best scaling using a line-search on the loss. Thus, we perform the operation $\theta_{\leftrightarrow}^K \leftarrow \gamma^* \theta_{\leftrightarrow}^K$, with the amplitude factor $\gamma^* \in \mathbb{R}$ defined as :

$$\gamma^* := \underset{\gamma \in [-L, L]}{\arg\min} \sum_i \mathcal{L}(f_{\theta \oplus \gamma \theta_{\leftrightarrow}^K}(\boldsymbol{x}_i), \boldsymbol{y}_i) \qquad \text{with } \gamma \theta_{\leftrightarrow}^{K^*} = (\gamma \alpha_k^*, \gamma \omega_k^*)_k^K \tag{14}$$

with L a positive constant[2]. More details can be found in Appendix E.3.2 and in Algorithm 1. With such an amplitude factor, one can measure the quality of the directions generated by TINY and Random by quantifying the maximal decrease of loss in these directions.

To better measure the impact of the initialisation method, and to distinguish it from the optimization process, we do not perform any gradient descent. This contrasts with the previous section where long training time after architecture growth was modifying the direction of the added neurons, dampening initialization impact with training time, especially as they were added with a small amplitude factor (cf Section E.3.1).

With these two modifications to the protocol of previous section, we obtain Figure 8. We see the crucial impact of TINY initialization compared to the Random one. Indeed, TINY reaches more than 17% accuracy just by adding neurons (without any further update), which accounts for about one quarter of the total accuracy with the full method (69% in Table 1 using in plus gradient descent). On the opposite, the Random initialization does not contribute to the accuracy through the growing process (just about 1%); this can be explained and quantified as follows. In the random setting, we model $\boldsymbol{v}(X)$ and $\boldsymbol{v}_{\text{goal}}(X)$ as independent variables where $\boldsymbol{v}_{\text{goal}} \sim \mathcal{N}(0_d, I_d \frac{1}{d})$ and $\boldsymbol{v}$ either $\sim \mathcal{N}(0_d, I_d \frac{1}{d})$ or $\sim \mathcal{U}[-\frac{1}{d}, \frac{1}{d}]$, d being the dimension of

[2]In practice, this is equivalent to a Glorot initialization

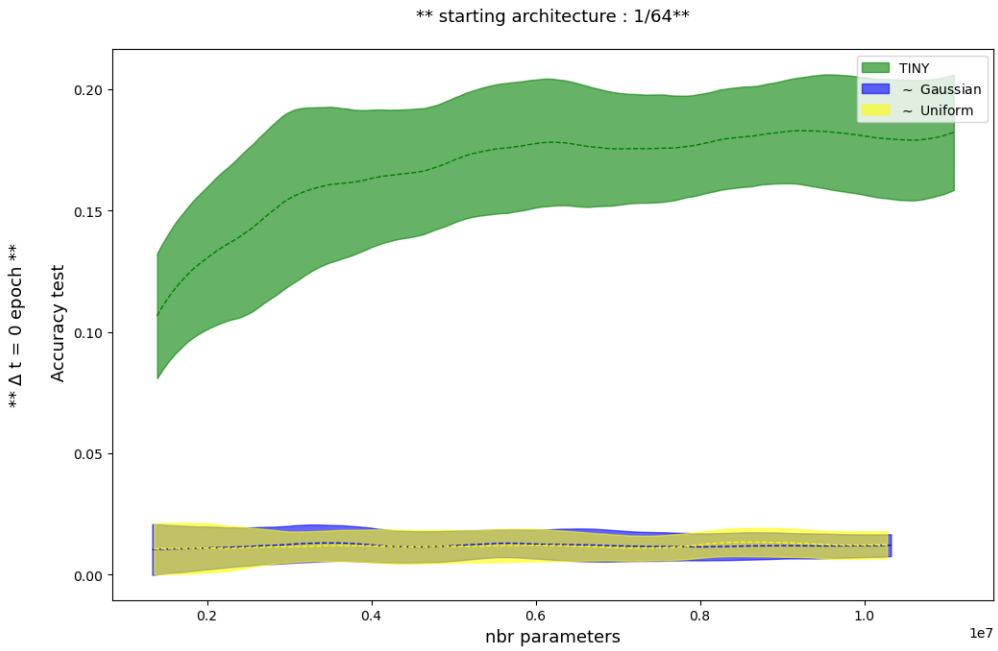

Figure 8: Test accuracy as a function of the number of parameters during architecture growth from $\text{ResNet}_{1/64}$ to ResNet_{18} on four independant runs.

$\boldsymbol{v}_{\text{goal}}$ and $\boldsymbol{v}$. From Equation 132, the scalar product $\langle \boldsymbol{V}(X), \boldsymbol{V}_{\text{goal}}(X) \rangle := \frac{1}{n} \sum_i \boldsymbol{v}_{\text{goal}}(\boldsymbol{x}_i)^T \boldsymbol{v}(\boldsymbol{x}_i)$ is a proxy of the expected decrease of loss after each architecture growth. This quantity can be approximated by its standard deviation, ie $\frac{1}{\sqrt{n\,d}}$, which makes the expected relative gain of loss (for a gradient step) of the order of magnitude of $\frac{1}{\sqrt{64}}$ for the first layer and $\frac{1}{\sqrt{512}}$ for the last layer when compared to the true gradient, and consequently when compared to TINY. Furthermore, one can take into account the effect of a line search over the random direction: in that case the expected relative loss gain is quadratic in the angle between the directions and therefore of the order of magnitude of $\frac{1}{64}$ or $\frac{1}{512}$ respectively (see Appendix D.3).

Note that the search interval of equation 14 for can be shrunk to $[0, L]$ with TINY initialization, as the first order development of the loss in Equation 132 is positive. This property is the direct consequence of the definition of $\boldsymbol{V}^*$ as the minimizer of the expressivity bottleneck equation 8. One can also note that we do not include GradMax in Figure 8, because its protocol initializes the on-going weights to zero ($\alpha_k \leftarrow 0$) and imposes a small norm on its out-going weights ($||\omega_k|| = \varepsilon$). Those two aspects make the amplitude factor γ^* meaningless and the impact of the new neurons initialization invisible without gradient descent.

The code will be available on git; for now the code is available as supplementary materials (zip file `TINY_code.zip`).

6 Conclusion

We provided the theoretical principles of TINY, a method to grow the architecture of a neural net while training it; starting from a very thin architecture, TINY adds neurons where needed and yields a fully trained architecture at the end. Our method relies on the functional gradient to find new directions that tackle the expressivity bottleneck, even for small networks, by expanding their space of parameters. This way, we combine in the same framework gradient descent and neural architecture search, that is, optimizing the network parameters and its architectures at the same time, and this, in a way that guarantees convergence to 0 training error, thus escaping expressivity bottlenecks indeed.

While transfer learning works well on ordinary tasks, for it to succeed, it needs to fine-tune and use large architectures at deployment in order to extract and manipulate common knowledge. Our method has the

advantage of being generic and could also produce smaller models as it adapts the architecture to a single task.

It is already instantiated for linear and convolutional layers and extension to self-attention mechanism (transformers) is part of future works. Although the common architectures consist of a succession of layers, a research direction is to develop tools handling general computational graphs (such as U-net, Inception, Dense-Net), which offers the possibility to let the architecture graph grow and bypass manual design.

Another possible development would be to study the statistical reliability of the TINY method, for instance using tools borrowed from random matrix theory. Indeed statistical tests can be applied to intermediate computations to obtain the new neurons. An interesting byproduct of this approach would be to define a threshold to select neurons found by 3.2, based on statistical significance.

References

Francis Bach. Breaking the curse of dimensionality with convex neural networks. *The Journal of Machine Learning Research*, 18(1):629–681, 2017.

Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. In *International Conference on Learning Representations*, 2017. URL `https://openreview.net/forum?id=S1c2cvqee`.

Kateryna Bashtova, Mathieu Causse, Cameron James, Florent Masmoudi, Mohamed Masmoudi, Houcine Turki, and Joshua Wolff. Application of the topological gradient to parsimonious neural networks. In *Computational Sciences and Artificial Intelligence in Industry*, pp. 47–61. Springer, 2022.

Pauline Bennet, Carola Doerr, Antoine Moreau, Jeremy Rapin, Fabien Teytaud, and Olivier Teytaud. Nevergrad: black-box optimization platform. *ACM SIGEVOlution*, 14(1):8–15, 2021.

Mathieu Causse, Cameron James, Mohamed Slim Masmoudi, and Houcine Turki. Parsimonious neural networks. In *Cesar Conference*, 2019. URL `https://www.cesar-conference.org/wp-content/uploads/2019/10/s5_p1_21_1330.pdf`.

Tianqi Chen, Ian Goodfellow, and Jonathon Shlens. Net2net: Accelerating learning via knowledge transfer. *arXiv preprint arXiv:1511.05641*, 2015.

G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

Bin Dai, Chen Zhu, Baining Guo, and David Wipf. Compressing neural networks using the variational information bottleneck. In *International Conference on Machine Learning*, pp. 1135–1144. PMLR, 2018.

Carl Eckart and Gale Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1(3):211–218, September 1936. ISSN 1860-0980. doi: 10.1007/BF02288367. URL `https://doi.org/10.1007/BF02288367`.

Utku Evci, Bart van Merrienboer, Thomas Unterthiner, Fabian Pedregosa, and Max Vladymyrov. Gradmax: Growing neural networks using gradient information. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=qjN4h_wwUO`.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Training pruned neural networks. *CoRR*, abs/1803.03635, 2018. URL `http://arxiv.org/abs/1803.03635`.

Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pp. 1189–1232, 2001.

Ariel Gordon, Elad Eban, Ofir Nachum, Bo Chen, Hao Wu, Tien-Ju Yang, and Edward Choi. Morphnet: Fast & simple resource-constrained structure learning of deep networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1586–1595, 2018.

Boris Hanin and David Rolnick. Complexity of linear regions in deep networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 2596–2604. PMLR, 09–15 Jun 2019a. URL https://proceedings.mlr.press/v97/hanin19a.html.

Boris Hanin and David Rolnick. Deep relu networks have surprisingly few activation patterns. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019b. URL https://proceedings.neurips.cc/paper/2019/file/9766527f2b5d3e95d4a733fcfb77bd7e-Paper.pdf.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

Arthur Jacot, Franck Gabriel, and Clement Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper/2018/file/5a4be1fa34e62bb8a6ec6b91d2462f5a-Paper.pdf.

Andrei N Kolmogorov. Three approaches to the quantitative definition ofinformation'. *Problems of information transmission*, 1(1):1–7, 1965.

Vladimir Koltchinskii. Rademacher penalties and structural risk minimization. *IEEE Transactions on Information Theory*, 47(5):1902–1914, 2001.

Jean-François Le Gall. *Intégration, Probabilités et Processus Aléatoires*. 2006.

Ming Li, Paul Vitányi, et al. *An introduction to Kolmogorov complexity and its applications*, volume 3. Springer, 2008.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=S1eYHoC5FX.

Zhengying Liu, Zhen Xu, Shangeth Rajaa, Meysam Madadi, Julio CS Jacques Junior, Sergio Escalera, Adrien Pavao, Sebastien Treguer, Wei-Wei Tu, and Isabelle Guyon. Towards automated deep learning: Analysis of the autodl challenge series 2019. In *NeurIPS 2019 Competition and Demonstration Track*, pp. 242–252. PMLR, 2020.

Christos Louizos, Karen Ullrich, and Max Welling. Bayesian compression for deep learning. *Advances in neural information processing systems*, 30, 2017.

Kaitlin Maile, Emmanuel Rachelson, Hervé Luga, and Dennis George Wilson. When, where, and how to add new neurons to ANNs. In *First Conference on Automated Machine Learning (Main Track)*, 2022. URL https://openreview.net/forum?id=SWOg-arIg9.

Hector Mendoza, Aaron Klein, Matthias Feurer, Jost Tobias Springenberg, and Frank Hutter. Towards automatically-tuned neural networks. In Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren (eds.), *Proceedings of the Workshop on Automatic Machine Learning*, volume 64 of *Proceedings of Machine Learning Research*, pp. 58–65, New York, New York, USA, 24 Jun 2016. PMLR. URL https://proceedings.mlr.press/v64/mendoza_towards_2016.html.

Risto Miikkulainen, Jason Zhi Liang, Elliot Meyerson, Aditya Rawal, Daniel Fink, Olivier Francon, Bala Raju, Hormoz Shahrzad, Arshak Navruzyan, Nigel Duffy, and Babak Hodjat. Evolving deep neural networks. *CoRR*, abs/1703.00548, 2017. URL http://arxiv.org/abs/1703.00548.

Geoffrey F. Miller, Peter M. Todd, and Shailesh U. Hegde. Designing neural networks using genetic algorithms. In *ICGA*, 1989.

Yann Ollivier. True asymptotic natural gradient optimization, 2017.

Allan Pinkus. Approximation theory of the mlp model in neural networks. *ACTA NUMERICA*, 8:143–195, 1999.

Maithra Raghu, Ben Poole, Jon Kleinberg, Surya Ganguli, and Jascha Sohl-Dickstein. On the expressive power of deep neural networks. In Doina Precup and Yee Whye Teh (eds.), *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 2847–2854. PMLR, 06–11 Aug 2017. URL `https://proceedings.mlr.press/v70/raghu17a.html`.

Thiago Serra, Christian Tjandraatmadja, and Srikumar Ramalingam. Bounding and counting linear regions of deep neural networks. In Jennifer Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 4558–4566. PMLR, 10–15 Jul 2018. URL `https://proceedings.mlr.press/v80/serra18b.html`.

Kenneth O Stanley, David B D'Ambrosio, and Jason Gauci. A hypercube-based encoding for evolving large-scale neural networks. *Artificial life*, 15(2):185–212, 2009.

Naftali Tishby and Noga Zaslavsky. Deep learning and the information bottleneck principle. *CoRR*, abs/1503.02406, 2015. URL `http://dblp.uni-trier.de/db/journals/corr/corr1503.html#TishbyZ15`.

V. N. Vapnik and A. Ya. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, 16(2):264–280, 1971. doi: 10.1137/1116025. URL `http://link.aip.org/link/?TPR/16/264/1`.

P. Wolinski, G. Charpiat, and O. Ollivier. Asymmetrical scaling layers for stable network pruning. *OpenReview Archive*, 2020.

Lemeng Wu, Dilin Wang, and Qiang Liu. Splitting steepest descent for growing neural architectures. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL `https://proceedings.neurips.cc/paper/2019/file/3a01fc0853ebeba94fde4d1cc6fb842a-Paper.pdf`.

Lemeng Wu, Bo Liu, Peter Stone, and Qiang Liu. Firefly neural architecture descent: a general approach for growing neural networks. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 22373–22383. Curran Associates, Inc., 2020. URL `https://proceedings.neurips.cc/paper/2020/file/fdbe012e2e11314b96402b32c0df26b7-Paper.pdf`.

Tien-Ju Yang, Andrew Howard, Bo Chen, Xiao Zhang, Alec Go, Mark Sandler, Vivienne Sze, and Hartwig Adam. Netadapt: Platform-aware neural network adaptation for mobile applications. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 285–300, 2018.

Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. In *International Conference on Learning Representations*, 2017. URL `https://openreview.net/forum?id=Sy8gdB9xx`.

Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning, 2016. URL `http://arxiv.org/abs/1611.01578`. cite arxiv:1611.01578.

Appendix outline

- Appendix A details the theoretical approach of TINY.

- Appendix B compares the theoretical approach of TINY with other approaches.
- Appendix C proves the propositions of the paper.
- Appendix E provides the hyper parameters for learning.
- Appendix F gives additional graphics associated to the result part.

For part B and C we use the trace scalar product and its associated norm. We note $\langle\ .\ ,\ .\ \rangle := \langle\ .\ ,\ .\ \rangle_{\mathrm{Tr}}$ and $||\ .\ || := ||\ .\ ||_{\mathrm{Tr}}$. One should remark that $||\ .\ || = ||\ .\ ||_{\mathrm{Tr}} = ||\ .\ ||_2$.

A Theoretical details for part 2

A.1 Functional gradient

The functional loss $\mathcal{L}$ is a functional that takes as input a function $f \in \mathcal{F}$ and outputs a real score:

$$\mathcal{L}:\ f \in \mathcal{F}\ \mapsto\ \mathcal{L}(f) = \mathop{\mathbb{E}}_{(\boldsymbol{x},\boldsymbol{y})\sim\mathcal{D}}\Big[\ell(f(\boldsymbol{x}),\boldsymbol{y})\Big]\ \in \mathbb{R}\ .$$

The function space $\mathcal{F}$ can typically be chosen to be $L_2(\mathbb{R}^p \to \mathbb{R}^d)$, which is a Hilbert space. The directional derivative (or Gateaux derivative, or Fréchet derivative) of functional $\mathcal{L}$ at function f in direction v is defined as:

$$D\mathcal{L}(f)(v) = \lim_{\varepsilon\to 0} \frac{\mathcal{L}(f+\varepsilon v) - \mathcal{L}(f)}{\varepsilon}$$

if it exists. Here v denotes any function in the Hilbert space $\mathcal{F}$ and stands for the direction in which we would like to update f, following an infinitesimal step (of size ε), resulting in a function $f + \varepsilon v$.

If this directional derivative exists in all possible directions $v \in \mathcal{F}$ and moreover is continuous in v, then the Riesz representation theorem implies that there exists a unique direction $v^* \in \mathcal{F}$ such that:

$$\forall v \in \mathcal{F},\ \ D\mathcal{L}(f)(v) = \langle v^*, v\rangle\ .$$

This direction v^* is named the *gradient* of the functional $\mathcal{L}$ at function f and is denoted by $\nabla_f\mathcal{L}(f)$.

Note that while the inner product $\langle\cdot,\cdot\rangle$ considered is usually the L_2 one, it is possible to consider other ones, such as Sobolev ones (e.g., H^1). The gradient $\nabla_f\mathcal{L}(F)$ depends on the chosen inner product and should consequently rather be denoted by $\nabla_f^{L_2}\mathcal{L}(f)$ for instance.

Note that continuous functions from $\mathbb{R}^p$ to $\mathbb{R}^d$, as well as C^∞ functions, are dense in $L_2(\mathbb{R}^p \to \mathbb{R}^d)$.

Let us now study properties specific to our loss design: $\mathcal{L}(f) = \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim\mathcal{D}}\Big[\ell(f(\boldsymbol{x}),\boldsymbol{y})\Big]$. Assuming sufficient ℓ-loss differentiability and integrability, we get, for any function update direction $v \in \mathcal{F}$ and infinitesimal step size $\varepsilon \in \mathbb{R}$:

$$\begin{aligned}\mathcal{L}(f+\varepsilon v) - \mathcal{L}(f) &= \mathop{\mathbb{E}}_{(\boldsymbol{x},\boldsymbol{y})\sim\mathcal{D}}\Big[\ell(f(\boldsymbol{x})+\varepsilon v(\boldsymbol{x}),\boldsymbol{y}) - \ell(f(\boldsymbol{x}),\boldsymbol{y})\Big]\\ &= \mathop{\mathbb{E}}_{(\boldsymbol{x},\boldsymbol{y})\sim\mathcal{D}}\Big[\nabla_{\boldsymbol{u}}\ell(\boldsymbol{u},\boldsymbol{y})\big|_{\boldsymbol{u}=f(\boldsymbol{x})}\cdot\varepsilon v(\boldsymbol{x}) + O(\varepsilon^2\|v(\boldsymbol{x})\|^2)\Big]\end{aligned}$$

using the usual gradient of function ℓ at point $(\boldsymbol{u} = f(\boldsymbol{x}), \boldsymbol{y})$ w.r.t. its first argument $\boldsymbol{u}$, with the standard Euclidean dot product $\cdot$ in $\mathbb{R}^p$. Then the directional derivative is:

$$D\mathcal{L}(f)(v) = \mathop{\mathbb{E}}_{(\boldsymbol{x},\boldsymbol{y})\sim\mathcal{D}}\Big[\nabla_{\boldsymbol{u}}\ell(\boldsymbol{u},\boldsymbol{y})\big|_{\boldsymbol{u}=f(\boldsymbol{x})}\cdot v(\boldsymbol{x})\Big] = \mathop{\mathbb{E}}_{\boldsymbol{x}\sim\mathcal{D}}\Big[\mathop{\mathbb{E}}_{\boldsymbol{y}\sim\mathcal{D}|\boldsymbol{x}}\Big[\nabla_{\boldsymbol{u}}\ell(\boldsymbol{u},\boldsymbol{y})\big|_{\boldsymbol{u}=f(\boldsymbol{x})}\Big]\cdot v(\boldsymbol{x})\Big]$$

and thus the functional gradient for the inner product $\langle v, v'\rangle_{\mathbb{E}} := \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\Big[v(\boldsymbol{x})\cdot v'(\boldsymbol{x})\Big]$ is the function:

$$\nabla_f^{\mathbb{E}}\mathcal{L}(f) : \boldsymbol{x} \mapsto \mathop{\mathbb{E}}_{\boldsymbol{y}\sim\mathcal{D}|\boldsymbol{x}}\Big[\nabla_{\boldsymbol{u}}\ell(\boldsymbol{u},\boldsymbol{y})\big|_{\boldsymbol{u}=f(\boldsymbol{x})}\Big]$$

which simplifies into:

$$\nabla_f^{\mathbb{E}} \mathcal{L}(f) : \boldsymbol{x} \mapsto \nabla_{\boldsymbol{u}} \ell(\boldsymbol{u}, \boldsymbol{y}(\boldsymbol{x}))\big|_{\boldsymbol{u}=f(\boldsymbol{x})}$$

if there is no ambiguity in the dataset, i.e. if for each $\boldsymbol{x}$ there is a unique $\boldsymbol{y}(\boldsymbol{x})$.

Note that by considering the $L_2(\mathbb{R}^p \to \mathbb{R}^d)$ inner product $\int v \cdot v'$ instead, one would respectively get:

$$\nabla_f^{L_2} \mathcal{L}(f) : \boldsymbol{x} \mapsto p_{\mathcal{D}}(\boldsymbol{x}) \mathop{\mathbb{E}}_{\boldsymbol{y} \sim \mathcal{D}|\boldsymbol{x}} \left[\nabla_{\boldsymbol{u}} \ell(\boldsymbol{u}, \boldsymbol{y})\big|_{\boldsymbol{u}=f(\boldsymbol{x})} \right]$$

and

$$\nabla_f^{L_2} \mathcal{L}(f) : \boldsymbol{x} \mapsto p_{\mathcal{D}}(\boldsymbol{x}) \, \nabla_{\boldsymbol{u}} \ell(\boldsymbol{u}, \boldsymbol{y}(\boldsymbol{x}))\big|_{\boldsymbol{u}=f(\boldsymbol{x})}$$

instead, where $p_{\mathcal{D}}(\boldsymbol{x})$ is the density of the dataset distribution at point $\boldsymbol{x}$. In practice one estimates such gradients using a minibatch of samples $(\boldsymbol{x}, \boldsymbol{y})$, obtained by picking uniformly at random within a finite dataset, and thus the formulas for the two inner products coincide (up to a constant factor).

A.2 Differentiation under the integral sign

Let X be an open subset of $\mathbb{R}$, and Ω be a measure space. Suppose $f : X \times \Omega \to \mathbb{R}$ satisfies the following conditions:

- $f(x, \omega)$ is a Lebesgue-integrable function of ω for each $x \in X$.
- For almost all $\omega \in \Omega$, the partial derivative $\frac{\partial}{\partial x} f$ of f according to x exists for all $x \in X$.
- There is an integrable function $\theta : \Omega \to \mathbb{R}$ such that $|\frac{\partial}{\partial x}(x, \omega)| \leq \theta(\omega)$ for all $x \in X$ and almost every $\omega \in \Omega$.

Then, for all $x \in X$,

$$\frac{\partial}{\partial x} \int_{\Omega} f(x, \omega) \, d\omega = \int_{\Omega} \frac{\partial}{\partial x} f(x, \omega) \, d\omega \tag{15}$$

See proof and details :Le Gall (2006).

A.3 Gradients and proximal point of view

Gradients with respect to standard variables such as vectors are defined the same way as functional gradients above: given a sufficiently smooth loss $\widetilde{\mathcal{L}} : \theta \in \Theta_{\mathcal{A}} \mapsto \widetilde{\mathcal{L}}(\theta) = \mathcal{L}(f_\theta) \in \mathbb{R}$, and an inner product $\cdot$ in the space $\Theta_{\mathcal{A}}$ of parameters θ, the gradient $\nabla_\theta \widetilde{\mathcal{L}}(\theta)$ is the unique vector $\boldsymbol{\tau} \in \Theta_{\mathcal{A}}$ such that:

$$\forall \delta\theta \in \Theta_{\mathcal{A}}, \quad \boldsymbol{\tau} \cdot \delta\theta = D_\theta \widetilde{\mathcal{L}}(\theta)(\delta\theta)$$

where $D_\theta \widetilde{\mathcal{L}}(\theta)(\delta\theta)$ is the directional derivative of $\widetilde{\mathcal{L}}$ at point θ in the direction $\delta\theta$, defined as in the previous section. This gradient depends on the inner product chosen, which can be highlighted by the following property. The opposite $-\nabla_\theta \widetilde{\mathcal{L}}(\theta)$ of the gradient is the unique solution of the problem:

$$\mathop{\arg\min}_{\delta\theta \in \Theta_{\mathcal{A}}} \left\{ D_\theta \widetilde{\mathcal{L}}(\theta)(\delta\theta) + \frac{1}{2} \|\delta\theta\|_P^2 \right\}$$

where $\| \quad \|_P$ is the norm associated to the chosen inner product. Changing the inner product obviously changes the way candidate directions $\delta\theta$ are penalized, leading to different gradients. This proximal formulation can be obtained as follows. For any $\delta\theta$, its distance to the gradient descent direction is:

$$\left\| \delta\theta - \left(-\nabla_\theta \widetilde{\mathcal{L}}(\theta) \right) \right\|^2 \;=\; \|\delta\theta\|^2 + 2\, \delta\theta \cdot \nabla_\theta \widetilde{\mathcal{L}}(\theta) + \left\| \nabla_\theta \widetilde{\mathcal{L}}(\theta) \right\|^2$$

$$= 2\left(\frac{1}{2}\left\|\delta\theta\right\|^2 + D_\theta\widetilde{\mathcal{L}}(\theta)(\delta\theta)\right) + K$$

where K does not depend on $\delta\theta$. For the above to hold, the inner product used has to be the one from which the norm is derived. By minimizing this expression with respect to $\delta\theta$, one obtains the desired property.

In our case of study, for the norm over the space $\Theta_{\mathcal{A}}$ of parameter variations, we consider a norm in the space of associated functional variations, i.e.:

$$\left\|\delta\theta\right\|_P \;:=\; \left\|\frac{\partial f_\theta}{\partial\theta}\,\delta\theta\right\|$$

which makes more sense from a physical point of view, as it is more intrinsic to the task to solve and depends as little as possible on the parameterization (i.e. on the architecture chosen). This results in a functional move that is the projection of the functional one to the set of possible moves given the architecture. On the opposite, the standard gradient (using Euclidean parameter norm $\|\delta\theta\|$ in parameter space) yields a functional move obtained not only by projecting the functional gradient but also by multiplying it by a matrix $\frac{\partial f_\theta}{\partial\theta}\frac{\partial f_\theta}{\partial\theta}^T$ which can be seen as a strong architecture bias over optimization directions.

We consider here that the loss $\mathcal{L}$ to be minimized is the real loss that the user wants to optimize, possibly including regularizers to avoid overfitting, and since the architecture is evolving during training, possibly to architectures far from usual manual design and never tested before, one cannot assume architecture bias to be desirable. We aim at getting rid of it in order to follow the functional gradient descent as closely as possible.

Searching for

$$\boldsymbol{v}^* \;=\; \underset{\boldsymbol{v}\in\mathcal{T}_{\mathcal{A}}}{\arg\min}\,\|\boldsymbol{v}-\boldsymbol{v}_{\text{goal}}\|^2 \;=\; \underset{\boldsymbol{v}\in\mathcal{T}_{\mathcal{A}}}{\arg\min}\left\{D\mathcal{L}(f)(\boldsymbol{v}) + \frac{1}{2}\|\boldsymbol{v}\|^2\right\} \tag{16}$$

or equivalently for:

$$\delta\theta^* \;=\; \underset{\delta\theta\in\Theta_{\mathcal{A}}}{\arg\min}\left\|\frac{\partial f_\theta}{\partial\theta}\,\delta\theta - \boldsymbol{v}_{\text{goal}}\right\|^2 \;=\; \underset{\delta\theta\in\Theta_{\mathcal{A}}}{\arg\min}\left\{D_\theta\mathcal{L}(f_\theta)(\delta\theta) + \frac{1}{2}\left\|\frac{\partial f_\theta}{\partial\theta}\,\delta\theta\right\|^2\right\} =: -\nabla_\theta^{\mathcal{T}_{\mathcal{A}}}\mathcal{L}(f_\theta) \tag{17}$$

then appears as a natural goal.

A.4 Example of expressivity bottleneck

Example. Suppose one tries to estimate the function $y = f_{\text{true}}(x) = 2\sin(x) + x$ with a linear model $f_{\text{predict}}(x) = ax + b$. Consider $(a, b) = (1, 0)$ and the square loss $\mathcal{L}$. For the dataset of inputs $(x_0, x_1, x_2, x_3) = (0, \frac{\pi}{2}, \pi, \frac{3\pi}{2})$, there exists no parameter update $(\delta a, \delta b)$ that would improve prediction at x_0, x_1, x_2 and x_3 simultaneously, as the space of linear functions $\{f : x \to ax + b \;\mid\; a, b \in \mathbb{R}\}$ is not expressive enough. To improve the prediction at x_0, x_1, x_2 **and** x_3, one should look for another, more expressive functional space such that for $i = 0, 1, 2, 3$ the functional update $\Delta f(x_i) := f^{t+1}(x_i) - f^t(x_i)$ goes into the same direction as the functional gradient $\boldsymbol{v}_{\text{goal}}(x_i) := -\nabla_{f(x_i)}\mathcal{L}(f(x_i), y_i) = -2(f(x_i) - y_i)$ where $y_i = f_{\text{true}}(x_i)$.

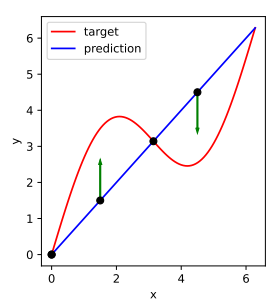

Figure 9: Linear interpolation

A.5 Problem formulation and choice of pre-activities

There are several ways to design the problem of adding neurons, which we discuss now, in order to explain our choice of the pre-activities to express expressivity bottlenecks.

Suppose one wishes to add K neurons $\theta_{\leftrightarrow}^K := (\boldsymbol{\alpha}_k, \boldsymbol{\omega}_k)_{k=1}^K$ to layer $l-1$, which impacts the activities $\boldsymbol{a}_l$ at the next layer, in order to improve its expressivity. These neurons could be chosen to have only nul weights,

or nul input weights $\boldsymbol{\alpha}_k$ and non-nul output weights $\boldsymbol{\omega}_k$, or the opposite, or both non-nul weights. Searching for the best neurons to add for each of these cases will produce different optimization problems.

Let us remind first that adding such K neurons with weights $\theta_{\leftrightarrow}^K := (\boldsymbol{\alpha}_k, \boldsymbol{\omega}_k)_{k=1}^K$ changes the activities $\boldsymbol{a}_l$ of the (next) layer by

$$\delta \boldsymbol{a}_l \quad = \quad \sum_{k=1}^K \boldsymbol{\omega}_k \; \sigma(\boldsymbol{\alpha}_k^T \boldsymbol{b}_{l-2}(\boldsymbol{x})) \tag{18}$$

Small weights approximation Under the hypothesis of small input weights α_k, the activity variation 18 can be approximated by:

$$\sigma'(0) \; \sum_{k=1}^K \boldsymbol{\omega}_k \boldsymbol{\alpha}_k^T \boldsymbol{b}_{l-2}(\boldsymbol{x}) \tag{19}$$

at first order in $\|\boldsymbol{\alpha}_k\|$. We will drop the constant $\sigma'(0)$ in the sequel.

This quantity is linear both in α_k and $\boldsymbol{\omega}_k$, therefore the first-order parameter-induced activity variations are easy to compute:

$$\boldsymbol{v}^l(\boldsymbol{x}, (\boldsymbol{\alpha}_k)_{k=1}^K) = \frac{\partial \boldsymbol{a}_l(\boldsymbol{x})}{\partial(\ (\boldsymbol{\alpha}_k)_{k=1}^K\ )}_{|(\boldsymbol{\alpha}_k)_{k=1}^K=0} (\boldsymbol{\alpha}_k)_{k=1}^K = \sum_{k=1}^K \boldsymbol{\omega}_k \boldsymbol{b}_{l-2}(\boldsymbol{x})^T \boldsymbol{\alpha}_k$$

$$\boldsymbol{v}^l(\boldsymbol{x}, (\boldsymbol{\omega}_k)_{k=1}^K) = \frac{\partial \boldsymbol{a}_l(\boldsymbol{x})}{\partial(\ (\boldsymbol{\omega}_k)_{k=1}^K\ )}_{|(\boldsymbol{\omega}_k)_{k=1}^K=0} (\boldsymbol{\omega}_k)_{k=1}^K = \sum_{k=1}^K \boldsymbol{\omega}_k \boldsymbol{b}_{l-2}(\boldsymbol{x})^T \boldsymbol{\alpha}_k$$

so with a slight abuse of notation we have:

$$\boldsymbol{v}^l(\boldsymbol{x}, \theta_{\leftrightarrow}^K) = \sum_{k=1}^K \boldsymbol{\omega}_k \boldsymbol{\alpha}_k^T \boldsymbol{b}_{l-2}(\boldsymbol{x})$$

Note also that technically the quantity above is first-order in $\boldsymbol{\alpha}_k$ and in $\boldsymbol{\omega}_k$ but second-order in the joint variable $\theta_{\leftrightarrow}^K = (\boldsymbol{\alpha}_k, \boldsymbol{\omega}_k)$.

Adding neurons with 0 weights (both input and output weights). In that case, one increases the number of neurons in the layer, but without changing the function (since only nul quantities are added) and also without changing the gradient with respect to the parameters, thus not improving expressivity. Indeed, the added quantity (Eq. 18) involves 0×0 multiplications, and consequently the derivative $\left.\frac{\partial \boldsymbol{a}_l(\boldsymbol{x})}{\partial \theta_{\leftrightarrow}^K}\right|_{\theta_{\leftrightarrow}^K=0}$ w.r.t. these new parameters, that is, $\boldsymbol{b}_{l-2}(\boldsymbol{x})^T \boldsymbol{\alpha}_k$ w.r.t. $\boldsymbol{\omega}_k$ and $\boldsymbol{\omega}_k \; \boldsymbol{b}_{l-2}(\boldsymbol{x})^T$ w.r.t. $\boldsymbol{a}_k$ is 0, as both $\boldsymbol{a}_k$ and $\boldsymbol{\omega}_k$ are 0.

Adding neurons with non-0 input weights and 0 output weights or the opposite. In these cases, the addition of neurons will not change the function (because of multiplications by 0), but just the gradient. One of the 2 gradients (w.r.t. $\boldsymbol{a}_k$ or w.r.t $\boldsymbol{\omega}_k$) will be non-0, as the variable that is 0 has non-0 derivatives.

The question is then how to pick the best non-0 variable, ($\boldsymbol{a}_k$ or $\boldsymbol{\omega}_k$) such that the added gradient will be the most useful. The problem can then be formulated similarly as what is done in the paper.

Adding neurons with small yet non-0 weights. In this case, both the function and its gradient will change when adding the neurons. Fortunately, Proposition 3.2 states that the best neurons to add in terms of expressivity (to get the gradient closer to the variation desired by the backpropagation) are also the best neurons to add to decrease the loss, i.e. the function change they will imply goes into the right direction.

For each family $(\boldsymbol{\omega}_k)_{k=1}^K$, the tangent space in $\boldsymbol{a}_l$ restricted to the family $(\boldsymbol{\alpha}_k)_{k=1}^K$, ie $\mathcal{T}_{\mathcal{A}}^{\boldsymbol{a}_l} := \{\frac{\partial \boldsymbol{a}_l}{\partial (\boldsymbol{\alpha}_k)_{k=1}^K}_{|(\boldsymbol{\alpha}_k)_{k=1}^K=0}(.)(\boldsymbol{\alpha}_k)_{k=1}^K | (\boldsymbol{\alpha}_k)_{k=1}^K \in \left(\mathbb{R}^{|\boldsymbol{b}_{l-2}(\boldsymbol{x})|}\right)^K\}$ varies with the family $(\boldsymbol{\omega}_k)_{k=1}^K$, ie $\mathcal{T}_{\mathcal{A}}^{a_l} := \mathcal{T}_{\mathcal{A}}^{a_l}((\boldsymbol{\omega}_k)_{k=1}^K)$. Optimizing w.r.t. the $\boldsymbol{\omega}_k$ is equivalent to search for the best tangent space for the $\boldsymbol{\alpha}_k$,

while symmetrically optimizing w.r.t. the $\boldsymbol{\alpha}_k$ is equivalent to find the best projection on the tangent space defined by the $\boldsymbol{\omega}_k$.

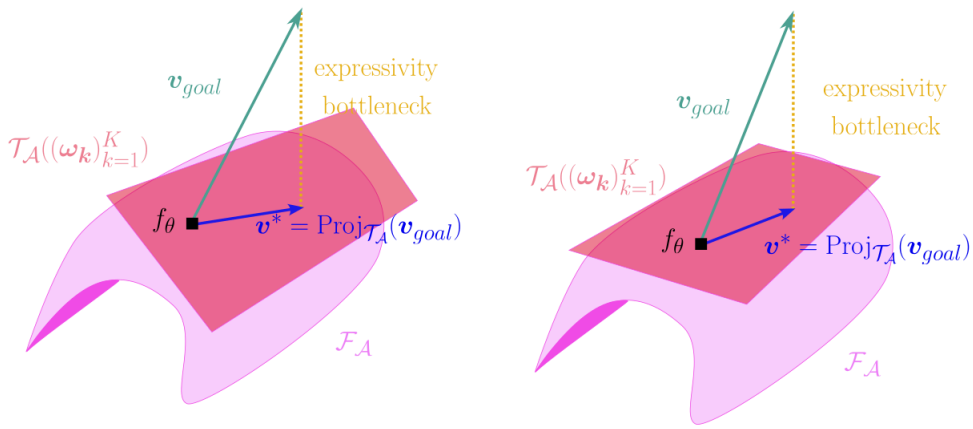

Figure 10: Changing the tangent space with different values of $(\boldsymbol{\omega}_k)_{k=1}^K$.

Pre-activities vs. post-activities. The space of pre-activities $\boldsymbol{a}_l$ is a natural space for this framework, as they are formed with linear operations and we compute first-order variation quantities. Considering the space of post-activities $\boldsymbol{b}_l = \sigma(\boldsymbol{a}_l)$ is also possible, though computing variations will be more complex. Indeed, without first-order approximation, the obtained problem is not manageable, because of the non-linear activation function σ added in front of all quantities (while in the case of pre-activations, quantity 18 is linear in $\boldsymbol{\omega}_k$ and thus does not require approximation in $\boldsymbol{\omega}_k$, which allow considering large $\boldsymbol{\omega}_k$), and, with first-order approximation, it would add the derivative of the activation function, taken at various locations $\sigma'(\boldsymbol{a}_l)$ (while in the previous case the derivatives of the activation function were always taken at 0).

A.6 Adding convolutional neurons

To add a convolutional neuron at layer $l-1$, one should add a kernel at layer $l-1$ and expand one dimension to all the kernels in layer l to match the new dimension of the post-activity.

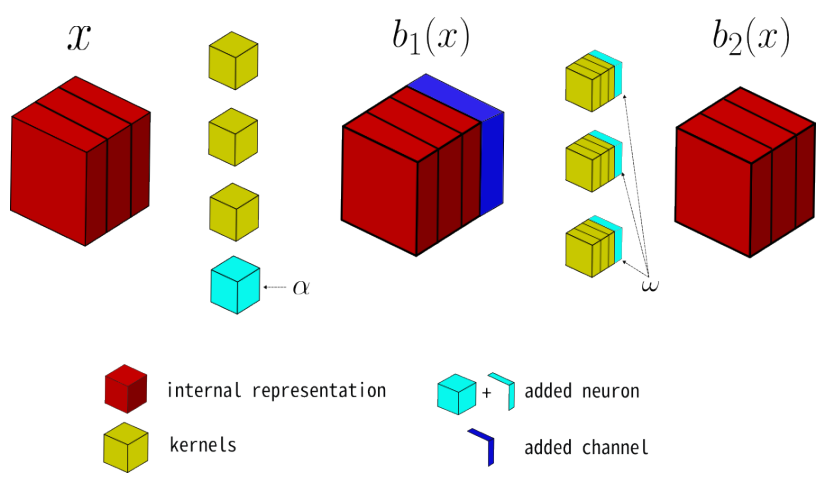

Figure 11: Adding one convolutional neuron at layer one for a input with tree channels.

B Theoretical comparison with other approaches

B.1 GradMax method

To facilitate reading we remove the layer index of each quantity, ie $\boldsymbol{b} := \boldsymbol{b}_{l-2}, \boldsymbol{B} := \boldsymbol{B}_{l-2}, \boldsymbol{V}_{\text{goal}} := \boldsymbol{V}_{\text{goal}}{}^{l}$ and $\boldsymbol{V}_{\text{goal}_{proj}} := \boldsymbol{V}_{\text{goal}}{}^{l}_{proj}$.

The theoretical approach of GradMax is to add neurons with zero fan-in and choose the fan-out which would decrease the loss as much as possible after one gradient step. We note $\boldsymbol{\Omega}$ the fan-out of such neurons and perform the addition at time t at layer l. After one gradient step, ie $t \to t+1$, the decrease of loss is :

$$\mathcal{L}^{t+1} \approx \mathcal{L}^{t} - ||\nabla_{\theta}\mathcal{L}||^2 - ||\nabla_{\Omega}\mathcal{L}||^2$$

The solution of GradMax as formulated in the paper Evci et al. (2022) is :

$$(\boldsymbol{\omega}_1^*, ..., \boldsymbol{\omega}_K^*) := \boldsymbol{\Omega}^* = \arg\max_{\boldsymbol{\Omega}} ||\nabla_{\boldsymbol{\Omega}}\mathcal{L}||^2 \qquad s.t. \;\; ||\boldsymbol{\Omega}||^2 \leq c \tag{20}$$

we remark that :

$$||\nabla_{\boldsymbol{\Omega}}\mathcal{L}||^2 = \left|\left|\sum_i \boldsymbol{b}(\boldsymbol{x}_i)\boldsymbol{v}_{\text{goal}}{}^T(\boldsymbol{x}_i)\boldsymbol{\Omega}\right|\right|^2 \tag{21}$$

$$= \left|\left|\boldsymbol{B}\boldsymbol{V}_{\text{goal}}{}^T\boldsymbol{\Omega}\right|\right|^2 \tag{22}$$

$$= \left|\left|\tilde{\boldsymbol{N}}\boldsymbol{\Omega}\right|\right|^2 \qquad \tilde{\boldsymbol{N}} := \boldsymbol{B}\boldsymbol{V}_{\text{goal}}{}^T \tag{23}$$

It follows that the fan-out of the neurons computed by GradMax are the solution of the problem that we shall prove later:

$$\boldsymbol{\Omega}^* := \arg\max_{\boldsymbol{\Omega}} \left|\left|\tilde{\boldsymbol{N}}\boldsymbol{\Omega}\right|\right| \quad s.t. \;\; ||\boldsymbol{\Omega}||^2 \leq c \tag{24}$$

To compare this optimization problem with TINY, we use the following proposition:

Proposition B.1.

$$\forall \boldsymbol{a} \in \mathbb{R}^d, \boldsymbol{M} \in \mathbb{R}^{j,d}, \; \exists c \in \mathbb{R} \; s.t \; \arg\min_{\boldsymbol{b}} ||\boldsymbol{a} - \boldsymbol{M}\boldsymbol{b}||^2 = \arg\max_{\boldsymbol{b}, \; ||\boldsymbol{M}\boldsymbol{b}||^2 \leq c} \langle \boldsymbol{a}, \boldsymbol{M}\boldsymbol{b}\rangle$$

Taking $\boldsymbol{V}_{\text{goal}}$ as $\boldsymbol{a}$ and $\boldsymbol{V}$ as $\boldsymbol{M}\boldsymbol{b}$, we can reformulate TINY optimization problem 8 as :

$$\boldsymbol{A}^*, \boldsymbol{\Omega}^* = \arg\max_{\boldsymbol{A},\boldsymbol{\Omega}} \langle \boldsymbol{V}(\boldsymbol{A}, \boldsymbol{\Omega}), \boldsymbol{V}_{\text{goal}_{proj}}\rangle \qquad s.t. \;\; ||\boldsymbol{V}(\boldsymbol{A}, \boldsymbol{\Omega})||^2 \leq c \tag{25}$$

We remark that

$$\langle \boldsymbol{V}(\boldsymbol{A}, \boldsymbol{\Omega}), \boldsymbol{V}_{\text{goal}_{proj}}\rangle = \langle \boldsymbol{\Omega}\boldsymbol{A}^T\boldsymbol{B}, \boldsymbol{V}_{\text{goal}_{proj}}\rangle \tag{26}$$

$$= \text{Tr}(\boldsymbol{B}^T\boldsymbol{A}\boldsymbol{\Omega}^T\boldsymbol{V}_{\text{goal}_{proj}}) \tag{27}$$

$$= \text{Tr}(\boldsymbol{A}\boldsymbol{\Omega}^T\boldsymbol{V}_{\text{goal}_{proj}}\boldsymbol{B}^T) \tag{28}$$

$$= \langle \boldsymbol{\Omega}\boldsymbol{A}^T, \boldsymbol{V}_{\text{goal}_{proj}}, \boldsymbol{B}^T\rangle \tag{29}$$

We perform the change of variable $\tilde{\boldsymbol{A}} := \boldsymbol{S}^{\frac{1}{2}}\boldsymbol{A}$ and re-write the constrain as :

$$||\boldsymbol{V}(\boldsymbol{A}, \boldsymbol{\Omega})||^2 = \left|\left|\boldsymbol{\Omega}\boldsymbol{A}^T\boldsymbol{B}\right|\right|^2 \tag{30}$$

$$= \text{Tr}(\boldsymbol{A}\boldsymbol{\Omega}^T\boldsymbol{\Omega}\boldsymbol{A}^T\boldsymbol{S}) \qquad \boldsymbol{S} = \boldsymbol{B}\boldsymbol{B}^T \tag{31}$$

$$= \left|\left|\boldsymbol{\Omega}(\mathbf{S}^{\frac{1}{2}}\boldsymbol{A})^T\right|\right| \tag{32}$$

$$= \left|\left|\boldsymbol{\Omega}\tilde{\boldsymbol{A}}^T\right|\right| \tag{33}$$

Then we define $\boldsymbol{N} := \boldsymbol{B}\boldsymbol{V}_{\text{goal}_{proj}}^T$. The initial scalar product is then :

$$\langle \boldsymbol{V}(\boldsymbol{A}, \boldsymbol{\Omega}), \boldsymbol{V}_{\text{goal}_{proj}} \rangle = \langle \boldsymbol{\Omega}\tilde{\boldsymbol{A}}^T \mathbf{S}^{-\frac{1}{2}}, \boldsymbol{N}^T \rangle = \langle \boldsymbol{\Omega}\tilde{\boldsymbol{A}}^T, \boldsymbol{N}^T \mathbf{S}^{-\frac{1}{2}} \rangle = \langle \tilde{\boldsymbol{A}}\boldsymbol{\Omega}^T, \mathbf{S}^{-\frac{1}{2}}\boldsymbol{N} \rangle \tag{34}$$

To maximise the scalar product, we choose $\tilde{\boldsymbol{A}}\boldsymbol{\Omega}^T = \mathbf{S}^{-\frac{1}{2}}\boldsymbol{N}$. A solution for $(\tilde{\boldsymbol{A}}, \boldsymbol{\Omega})$ is the (left, right) eigenvectors of the matrix $\mathbf{S}^{-\frac{1}{2}}\boldsymbol{N}$. It implies that :

$$\boldsymbol{\Omega}^* := \arg\max_{\boldsymbol{\Omega}} \left|\left|\mathbf{S}^{-\frac{1}{2}}\boldsymbol{N}\boldsymbol{\Omega}\right|\right|^2 \quad s.t. \, ||\boldsymbol{\Omega}|| \leq \tilde{c} \tag{35}$$

One can note three differences between those optimization problems:

- First, the matrix $\tilde{\boldsymbol{N}}$ is not defined using the projection of the desired update $\boldsymbol{V}_{\text{goal}_{proj}}^{l+1}$. As a consequence, GradMax does not take into account redundancy, and on the opposite will actually try to add new neurons that are as redundant as possible with the part of the goal update that is already feasible with already-existing neurons.
- Second, the constraint lies in the weight space for GradMax method while it lies in the pre-activation space in our case. The difference is that GradMax relies on the Euclidean metric in the space of parameters, which arguably offers less meaning that the Euclidean metric in the space of activities. Essentially this is the same difference as between the standard L2 gradient w.r.t. parameters and the natural gradient, which takes care of parameter redundancy and measures all quantities in the output space in order to be independent from the parameterization. In practice we do observe that the "natural" gradient direction improves the loss better than the usual L2 gradient.
- Third, our fan-in weights are not set to 0 but directly to their optimal values (at first order).

We now prove the proposition B.1

Proof. Indeed,

$$\arg\min_{\boldsymbol{b}} ||\boldsymbol{a} - \boldsymbol{M}\boldsymbol{b}||^2 = \arg\min_{\boldsymbol{b}} ||\boldsymbol{M}\boldsymbol{b}||^2 - \langle \boldsymbol{a}, \boldsymbol{M}\boldsymbol{b} \rangle \tag{36}$$

$$= \arg\min_{\boldsymbol{b}} ||\boldsymbol{M}\boldsymbol{b}|| \left(||\boldsymbol{M}\boldsymbol{b}|| - \left\langle \boldsymbol{a}, \frac{\boldsymbol{M}\boldsymbol{b}}{||\boldsymbol{M}\boldsymbol{b}||} \right\rangle \right) \tag{37}$$

$$= \arg\min_{c} \arg\min_{||\boldsymbol{M}\boldsymbol{b}||=c, \, \boldsymbol{u}=\frac{\boldsymbol{M}\boldsymbol{b}}{||\boldsymbol{M}\boldsymbol{b}||}} c - \langle \boldsymbol{a}, \boldsymbol{u} \rangle \tag{38}$$

With the convention that $\frac{0}{||0||} = 0$. □

B.2 NORTH Preactivation

In paper Maile et al. (2022), fan-out weights are initialized to 0 while fan-in weights are initialized as $\boldsymbol{\alpha}_i = \boldsymbol{S}^{-1}\boldsymbol{B}_{l-1}\mathbf{V}_{\mathbf{Z}_{l-1}}^T r_i$ where r_i is a random vector and $\mathbf{V}_{\mathbf{Z}_{l-1}} \in \mathcal{M}(|Ker(\boldsymbol{B}_{l-1}^T)|, |\boldsymbol{b}_{l-2}(\boldsymbol{x})|)$ is a matrix consisting of the orthogonal vectors of the kernel of pre-activations $\boldsymbol{B}_{l-1}$, i.e $\{z \mid \boldsymbol{B}_{l-1}^T z = 0\}$. In our paper fan-in weights are initialized as $\boldsymbol{\alpha}_i = \boldsymbol{S}^{-1}\boldsymbol{B}_{l-2}\boldsymbol{V}_{\text{goal}_{proj}}^T \boldsymbol{v}_i = \boldsymbol{S}^{-1}\boldsymbol{B}_{l-2}\boldsymbol{V}_{\text{goal}}{}^T \mathbf{V}_{\mathbf{Z}_{l-1}}\mathbf{V}_{\mathbf{Z}_{l-1}}^T \boldsymbol{v}_i$, where the $\boldsymbol{v}_i$ are right eigenvectors of the matrix $\boldsymbol{S}^{-\frac{1}{2}}\boldsymbol{N}$.

The main difference is thus that we use the backpropagation to find the best $\boldsymbol{v}_i$ or r_i directly, while the NORTH approach tries random directions r_i to explore the space of possible neuron additions.

C Proofs of Part 3

C.1 Proposition 3.1

Denoting by $\boldsymbol{M}^+$ the generalized (pseudo-)inverse of $\boldsymbol{M}$, we have:

$$\delta\boldsymbol{W}_l^* = \frac{1}{n}\boldsymbol{V}_{\text{goal}}{}^l \boldsymbol{B}_{l-1}^T \big(\frac{1}{n}\boldsymbol{B}_{l-1}\boldsymbol{B}_{l-1}^T\big)^+ \text{ and } \boldsymbol{V}_0^l = \frac{1}{n}\boldsymbol{V}_{\text{goal}}{}^l \boldsymbol{B}_{l-1}^T \big(\frac{1}{n}\boldsymbol{B}_{l-1}\boldsymbol{B}_{l-1}^T\big)^+ \boldsymbol{B}_{l-1}$$

Proof
Consider the function

$$g(\delta \boldsymbol{W}) = ||\boldsymbol{V}_{\text{goal}}{}^{l} - \delta \boldsymbol{W} \boldsymbol{B}_{l-1}||^2 \tag{39}$$

then:

$$\begin{aligned}
g(\delta \boldsymbol{W} + \boldsymbol{H}) &= ||\boldsymbol{V}_{\text{goal}}{}^{l} - \delta \boldsymbol{W} \boldsymbol{B}_{l-1} - \boldsymbol{H} \boldsymbol{B}_{l-1}||^2 && (40)\\
&= g(\delta \boldsymbol{W}) - 2\langle \boldsymbol{V}_{\text{goal}}{}^{l} - \delta \boldsymbol{W} \boldsymbol{B}_{l-1}, \boldsymbol{H} \boldsymbol{B}_{l-1}\rangle + o(||\boldsymbol{H}||) && (41)\\
&= g(\delta \boldsymbol{W}) - 2\,\text{Tr}\left(\left(\boldsymbol{V}_{\text{goal}}{}^{l} - \delta \boldsymbol{W} \boldsymbol{B}_{l-1}\right)^T \boldsymbol{H} \boldsymbol{B}_{l-1}\right) + o(||\boldsymbol{H}||) && (42)\\
&= g(\delta \boldsymbol{W}) - 2\,\text{Tr}\left(\boldsymbol{B}_{l-1}\left(\boldsymbol{V}_{\text{goal}}{}^{l} - \delta \boldsymbol{W} \boldsymbol{B}_{l-1}\right)^T \boldsymbol{H}\right) + o(||\boldsymbol{H}||) && (43)\\
&= g(\delta \boldsymbol{W}) - 2\langle \left(\boldsymbol{V}_{\text{goal}}{}^{l} - \delta \boldsymbol{W} \boldsymbol{B}_{l-1}\right) \boldsymbol{B}_{l-1}^T, \boldsymbol{H}\rangle + o(||\boldsymbol{H}||) && (44)
\end{aligned}$$

By identification $\nabla_{\delta \boldsymbol{W}} g(\delta \boldsymbol{W}) = -2\left(\boldsymbol{V}_{\text{goal}}{}^{l} - \delta \boldsymbol{W} \boldsymbol{B}_{l-1}\right) \boldsymbol{B}_{l-1}^T$, and thus:

$$\nabla_{\delta \boldsymbol{W}} g(\delta \boldsymbol{W}) = 0 \implies \boldsymbol{V}_{\text{goal}}{}^{l} \boldsymbol{B}_{l-1}^T = \delta \boldsymbol{W} \boldsymbol{B}_{l-1} \boldsymbol{B}_{l-1}^T$$

Using that g is convex and the definition of the generalized inverse, we get:

$$\delta \boldsymbol{W}_l^* = \frac{1}{n} \boldsymbol{V}_{\text{goal}}{}^{l} \boldsymbol{B}_{l-1}^T \left(\frac{1}{n} \boldsymbol{B}_{l-1} \boldsymbol{B}_{l-1}^T\right)^+$$

as one solution. **For convolutional layer**, we definied as b_i^c the input associated to the activation $a_l(X_i) \in \mathbb{R}^{k,p,p}$, such that for a convolution layer with one output channel, noted with parameter $\boldsymbol{W}$, we have :

$$Conv(a_l(X_i)) = \boldsymbol{B}_i^c \; vect(\boldsymbol{W}) \tag{45}$$

Example : considering the kernel of $Conv$ to be $(2, 2)$, then :

$$\boldsymbol{b}_i^k = \begin{pmatrix} b_i^{1,k} & b_i^{2,k} & . & b_i^{p,k} \\ b_i^{p+1,k} & b_i^{p+2,k} & . & b_i^{2p,k} \\ . & . & . & . \\ b_i^{p(p-1)+1,k} & . & . & b_i^{p^2,k} \end{pmatrix} \tag{46}$$

$$\boldsymbol{B}_i^c = \begin{pmatrix} b_i^{1,1} & b_i^{2,1} & b_i^{p+1,1} & b_i^{p+2,1} & b_i^{1,2} & b_i^{2,2} & b_i^{p+1,2} & b_i^{p+2,2} & b_i^{1,3} & . \\ b_i^{2,1} & b_i^{3,1} & b_i^{p+2,1} & b_i^{p+3,1} & b_i^{2,2} & b_i^{3,2} & b_i^{p+2,2} & b_i^{p+3,2} & b_i^{2,3} & . \\ . & . & . & . & . & . & . & . & . & . \end{pmatrix}$$

Then the function to minimize is

$$g(\delta \boldsymbol{W}) = ||\boldsymbol{V}_{\text{goal}}{}^{l} - \boldsymbol{B}^c \delta \boldsymbol{W}||^2$$

where $\boldsymbol{B}^c := \begin{pmatrix} \boldsymbol{B}_1^c & ... & \boldsymbol{B}_n^c \end{pmatrix}$ and $\delta \boldsymbol{W}$ is the concatenation over the output channels. □

C.2 Proposition 3.2

We define the matrices $\boldsymbol{N} := \frac{1}{n} \boldsymbol{B}_{l-2} \left(\boldsymbol{V}_{\text{goal}}{}_{proj}^{l}\right)^T$ and $\boldsymbol{S} := \frac{1}{n} \boldsymbol{B}_{l-2} \boldsymbol{B}_{l-2}^T$. Let us denote its SVD by $\boldsymbol{S} = \boldsymbol{U} \Sigma \boldsymbol{U}^T$, and note $\boldsymbol{S}^{-\frac{1}{2}} := \boldsymbol{U} \sqrt{\Sigma}^{-1} \boldsymbol{U}^T$ and consider the SVD of the matrix $\boldsymbol{S}^{-\frac{1}{2}} \boldsymbol{N} = \sum_{k=1}^{R} \lambda_k \boldsymbol{u}_k \boldsymbol{v}_k^T$ with $\lambda_1 \geq ... \geq \lambda_R \geq 0$, where R is the rank of the matrix $\boldsymbol{N}$. Then:

Proposition C.1 (3.2)**.** *The solution of (7) can be written as:*

- *optimal number of neurons:* $K^* = R$
- *their optimal weights:* $(\boldsymbol{\alpha}_k^*, \boldsymbol{\omega}_k^*) = (\sqrt{\lambda_k} S^{-\frac{1}{2}} \boldsymbol{u}_k, \sqrt{\lambda_k} \boldsymbol{v}_k)$ *for* $k = 1, ..., R$.

Moreover for any number of neurons $K \leqslant R$*, and associated scaled weights* $\theta_{\leftrightarrow}^{K,*}$*, the expressivity gain and the first order in* η *of the loss improvement due to the addition of these* K *neurons are equal and can be quantified very simply as a function of the eigenvalues* λ_k*:*

$$\Psi^l_{\theta \oplus \theta_{\leftrightarrow}^{K,*}} = \Psi^l_\theta - \sum_{k=1}^{K} \lambda_k^2$$

With an inequality (≤) for convolutional layer.

Proof.
To facilitate reading we remove the layer index of each quantity, ie $\boldsymbol{B} := \boldsymbol{B}_{l-2}, \boldsymbol{V}^l(\boldsymbol{A}, \boldsymbol{\Omega}) := \boldsymbol{V}(\boldsymbol{A}, \boldsymbol{\Omega})$ and $\boldsymbol{V}_{\text{goal}_{proj}}^l := \boldsymbol{V}_{\text{goal}_{proj}}$. We fix n and $\boldsymbol{x}_1, ..., \boldsymbol{x}_n$ on which we solve the expressivity bottleneck formula.

To solve this problem, we consider the input of the incomming connexions $\boldsymbol{B}$ and the desired change in the output of the outgoing connexions $\boldsymbol{V}_{\text{goal}_{proj}}$. Hence if we note $L(\boldsymbol{A})$ and $L(\boldsymbol{\Omega})$ the additional connexions of the expanded representation and σ the non linearity, we optimize the following proxy problem:

$$\underset{\boldsymbol{A},\boldsymbol{\Omega}}{\arg\min} \frac{1}{n} \left|\left|(L(\boldsymbol{\Omega}) \circ \sigma \circ L(\boldsymbol{A}))(\boldsymbol{B}) - \boldsymbol{V}_{\text{goal}_{proj}}\right|\right|_{\text{Tr}} \tag{47}$$

We solve this problem at first order by linearizing the non linearity σ. We denote $\mathbf{Lin}_{(a,b)}(W)$ the fully connected layer with input size a, output size b and weight matrix $\boldsymbol{W}$. We also note $C[+1]$ and $C[-1]$ the layer width at layer $l+1$ and $l+1$ with the convention that C[0] is the dimension of the input X. With those notations, for fully connected layers, we have for the additions of K neurons:

$$\underset{\boldsymbol{A},\boldsymbol{\Omega}}{\arg\min} \frac{1}{n} \left|\left|\mathbf{Lin}_{(C[+1],K)}(\boldsymbol{\Omega})(\mathbf{Lin}_{(K,C[-1])}(\boldsymbol{A})(\boldsymbol{B})) - \boldsymbol{V}_{\text{goal}_{proj}}\right|\right|_2 \tag{48}$$

With the same notations, for convolutional layers, we have for the additions of K intermediate channels:

$$\underset{\boldsymbol{A},\boldsymbol{\Omega}}{\arg\min} \frac{1}{n} \left|\left|\mathbf{Conv}_{(C[+1],K)}(\boldsymbol{\Omega})(\mathbf{Conv}_{(K,C[-1])}(\boldsymbol{A})(\boldsymbol{B})) - \boldsymbol{V}_{\text{goal}_{proj}}\right|\right|_2 \tag{49}$$

If we note $\boldsymbol{V}(\boldsymbol{A}, \boldsymbol{\Omega})$ the result of $\boldsymbol{B}$ affter applying the layers parametrized by $\boldsymbol{A}$ and $\boldsymbol{\Omega}$, in both cases we aim to optimize:

$$\underset{\boldsymbol{A},\boldsymbol{\Omega}}{\arg\min} \frac{1}{n} \left|\left|\boldsymbol{V}(\boldsymbol{A}, \boldsymbol{\Omega}) - \boldsymbol{V}_{\text{goal}_{proj}}\right|\right|_{\text{Tr}} \tag{50}$$

First we will transform the resolution of the problem in solving the following optimization problem:

$$\underset{\boldsymbol{A},\boldsymbol{\Omega}}{\arg\min} \left|\left|\mathbf{S}^{\frac{1}{2}} \boldsymbol{A}\boldsymbol{\Omega} - \mathbf{S}^{-\frac{1}{2}} \boldsymbol{N}\right|\right|_2 \tag{51}$$

where $\boldsymbol{S}$ depends of $\boldsymbol{B}$ and $\boldsymbol{N}$ of $\boldsymbol{B}$ and $\boldsymbol{V}_{\text{goal}_{proj}}$.

If we note $\boldsymbol{S} = \boldsymbol{U}\Lambda\boldsymbol{U}^\top$ the SVD of $\boldsymbol{S}$, we define the square root of $\boldsymbol{S}$ as $\mathbf{S}^{\frac{1}{2}} := \boldsymbol{U}\sqrt{\Lambda}\boldsymbol{U}^\top$ and $\mathbf{S}^{-\frac{1}{2}} := \boldsymbol{U}\sqrt{\Lambda^{-1}}\boldsymbol{U}^\top$ with the convention $0^{-1} = 0$.

Lemma C.1. *For* $\boldsymbol{S} \in \mathbb{R}(s,s), \boldsymbol{N} \in \mathbb{R}(s,t), \boldsymbol{A} \in \mathbb{R}(s,k), \boldsymbol{B} \in \mathbb{R}(k,t)$.
If $\boldsymbol{N} = \mathbf{S}^{\frac{1}{2}}\mathbf{S}^{-\frac{1}{2}}\boldsymbol{N}$*, we have:*

$$\langle \boldsymbol{AB}, \boldsymbol{SAB} \rangle - 2\langle \boldsymbol{N}, \boldsymbol{AB} \rangle = \left|\left|\mathbf{S}^{\frac{1}{2}}\boldsymbol{AB} - \mathbf{S}^{-\frac{1}{2}}\boldsymbol{N}\right|\right|^2 - \left|\left|\mathbf{S}^{-\frac{1}{2}}\boldsymbol{N}\right|\right|^2 \tag{52}$$

Proof.

- For the first term we have:

$$\langle \boldsymbol{AB}, \boldsymbol{SAB} \rangle = \left\langle \boldsymbol{AB}, \mathbf{S}^{\frac{1}{2}}\mathbf{S}^{\frac{1}{2}}\boldsymbol{AB} \right\rangle \tag{53}$$
$$= \left\langle \mathbf{S}^{\frac{1}{2}}\boldsymbol{AB}, \mathbf{S}^{\frac{1}{2}}\boldsymbol{AB} \right\rangle \tag{54}$$
$$= \left|\left|\mathbf{S}^{\frac{1}{2}}\boldsymbol{AB}\right|\right|^2 \tag{55}$$

- For the second term we have:

$$\langle \boldsymbol{N}, \boldsymbol{AB} \rangle = \left\langle \mathbf{S}^{\frac{1}{2}}\mathbf{S}^{-\frac{1}{2}}\boldsymbol{N}, \boldsymbol{AB} \right\rangle \tag{56}$$
$$= \left\langle \mathbf{S}^{-\frac{1}{2}}\boldsymbol{N}, \mathbf{S}^{\frac{1}{2}}\boldsymbol{AB} \right\rangle \tag{57}$$

Hence we have that:

$$\langle \boldsymbol{AB}, \boldsymbol{SAB} \rangle - 2\langle \boldsymbol{N}, \boldsymbol{AB} \rangle = \left|\left|\mathbf{S}^{\frac{1}{2}}\boldsymbol{AB}\right|\right|^2 - 2\langle \boldsymbol{N}, \boldsymbol{AB} \rangle + \left|\left|\mathbf{S}^{-\frac{1}{2}}\boldsymbol{N}\right|\right|^2 - \left|\left|\mathbf{S}^{-\frac{1}{2}}\boldsymbol{N}\right|\right|^2 \tag{58}$$
$$= \left|\left|\mathbf{S}^{\frac{1}{2}}\boldsymbol{AB} - \mathbf{S}^{-\frac{1}{2}}\boldsymbol{N}\right|\right|^2 - \left|\left|\mathbf{S}^{-\frac{1}{2}}\boldsymbol{N}\right|\right|^2 \tag{59}$$

□

C.2.1 Fully connected layers

For a fully connected layer, we have

$$\boldsymbol{V}(\boldsymbol{A}, \boldsymbol{\Omega}) = \mathbf{Lin}_{(C[+1],K)}(\boldsymbol{\Omega})(\mathbf{Lin}_{(K,C[-1])}(\boldsymbol{A})(\boldsymbol{B})) = \boldsymbol{BA\Omega} \tag{60}$$

We will use the following lemma to get the desired result:
Lemma C.2. *Let* $\boldsymbol{Y} \in \mathbb{R}(n,t), \boldsymbol{X} \in \mathbb{R}(n,s)$ *and* $\boldsymbol{S} := \boldsymbol{X}^\top \boldsymbol{X} \in \mathbb{R}(s,s)$.

$$\mathbf{S}^{\frac{1}{2}}\mathbf{S}^{-\frac{1}{2}}\boldsymbol{X}^T\boldsymbol{Y} = \boldsymbol{X}^T\boldsymbol{Y} \tag{61}$$

Proof. Let decompose $\boldsymbol{Y}$ on $\mathrm{Im}(\boldsymbol{X}) \oplus_\perp \ker(\boldsymbol{X}^\top)$: $\boldsymbol{Y} = \boldsymbol{X}I + \boldsymbol{K}$.

$$\boldsymbol{X}^T\boldsymbol{Y} = \boldsymbol{X}^T\boldsymbol{X}I + \boldsymbol{X}^T K = \boldsymbol{X}^T\boldsymbol{X}I = \boldsymbol{S}I$$

Hence $\boldsymbol{X}^T\boldsymbol{Y} \in \mathrm{Im}(\boldsymbol{S})$, hence as $\boldsymbol{S}_{|\,\mathrm{Im}(\boldsymbol{S})}$ is invertible, we have: $\mathbf{S}^{-\frac{1}{2}}\mathbf{S}^{\frac{1}{2}}\boldsymbol{X}^T\boldsymbol{Y} = \boldsymbol{X}^T\boldsymbol{Y}$. □

Lemma C.3. *Let* $\boldsymbol{Y} \in \mathbb{R}(n,t), \boldsymbol{X} \in \mathbb{R}(n,s),\ k \leq \min(s,t), \boldsymbol{A} \in \mathbb{R}(s,k), \boldsymbol{B} \in \mathbb{R}(k,t)$.
We define:

$$\boldsymbol{S} := \frac{1}{n}\boldsymbol{X}^\top\boldsymbol{X} \in \mathbb{R}(s,s) \tag{62}$$
$$\boldsymbol{N} := \frac{1}{n}\boldsymbol{X}^\top\boldsymbol{Y} \in \mathbb{R}(s,t) \tag{63}$$

$$\frac{1}{n}\,||\boldsymbol{Y} - \boldsymbol{XAB}||^2 = \left|\left|\mathbf{S}^{\frac{1}{2}}\boldsymbol{AB} - \mathbf{S}^{-\frac{1}{2}}\boldsymbol{X}^\top\boldsymbol{Y}\right|\right|^2 - \left|\left|\mathbf{S}^{-\frac{1}{2}}\boldsymbol{X}^\top\boldsymbol{Y}\right|\right|^2 + ||\boldsymbol{Y}||^2 \tag{64}$$

Proof. By developing the scalar product we get:

$$\frac{1}{n}\left|\left|\boldsymbol{Y}-\boldsymbol{X}\boldsymbol{A}\boldsymbol{B}\right|\right|^2=\frac{1}{n}\left|\left|\boldsymbol{Y}\right|\right|^2-2\left\langle\boldsymbol{Y},\frac{1}{n}\boldsymbol{X}\boldsymbol{A}\boldsymbol{B}\right\rangle+\frac{1}{n}\left|\left|\boldsymbol{X}\boldsymbol{A}\boldsymbol{B}\right|\right|^2 \tag{65}$$

$$=\frac{1}{n}\left|\left|\boldsymbol{Y}\right|\right|^2-2\left\langle\boldsymbol{Y},\frac{1}{n}\boldsymbol{X}\boldsymbol{A}\boldsymbol{B}\right\rangle+\frac{1}{n}\left\langle\boldsymbol{X}\boldsymbol{A}\boldsymbol{B},\boldsymbol{X}\boldsymbol{A}\boldsymbol{B}\right\rangle \tag{66}$$

$$=\frac{1}{n}\left|\left|\boldsymbol{Y}\right|\right|^2-2\left\langle\frac{1}{n}\boldsymbol{X}^T\boldsymbol{Y},\boldsymbol{A}\boldsymbol{B}\right\rangle+\left\langle\boldsymbol{A}\boldsymbol{B},\frac{1}{n}\boldsymbol{X}^\top\boldsymbol{X}\boldsymbol{A}\boldsymbol{B}\right\rangle \tag{67}$$

$$\tag{68}$$

Using theorem C.2 we can apply theorem C.1 and immediately get the result. □

Hence using theorem C.3 we have:

$$\begin{gathered}\frac{1}{n}\left|\left|\mathbf{Lin}_{(C[+1],K)}(\boldsymbol{\Omega})(\mathbf{Lin}_{(K,C[-1])}(\boldsymbol{A})(\boldsymbol{B}))-\boldsymbol{V}_{\text{goal}_{proj}}\right|\right|^2\\ =\\ \left|\left|\mathbf{S}^{\frac{1}{2}}\boldsymbol{A}\boldsymbol{\Omega}-\mathbf{S}^{-\frac{1}{2}}\boldsymbol{N}\right|\right|^2-\left|\left|\mathbf{S}^{-\frac{1}{2}}\boldsymbol{N}\right|\right|^2+\frac{1}{n}\left|\left|\boldsymbol{V}_{\text{goal}_{proj}}\right|\right|^2\end{gathered} \tag{69}$$

With:

$$\boldsymbol{S}:=\frac{1}{n}\boldsymbol{B}^\top\boldsymbol{B}\in\mathbb{R}(C[-1],C[-1])$$

$$\boldsymbol{N}:=\frac{1}{n}\boldsymbol{B}^\top\boldsymbol{V}_{\text{goal}_{proj}}\in\mathbb{R}(C[-1],C[+1])$$

C.2.2 Convolutional connected layers

We have $\boldsymbol{A}\in\mathbb{R}(K,C[-1],d,d)$ and $\boldsymbol{\Omega}\in\mathbb{R}(C[+1],K,d[+1],d[+1])$ where $d,d[+1]$ is the kernel size at l and $l+1$. We note $\boldsymbol{A}_F\in\mathbb{R}(K,C[-1]dd)$ the flatten version of $\boldsymbol{A}$ and $\boldsymbol{A}_k:=\boldsymbol{A}_F[k,:]$. We also note $\boldsymbol{\Omega}_{m,k}\in\mathbb{R}(d[+1]d[+1])$ the flatten version of $\boldsymbol{\Omega}[m,k,:,:]$. Using this we define $\boldsymbol{\Omega}_k:=\begin{pmatrix}\boldsymbol{\Omega}_{k,1} & \cdots & \boldsymbol{\Omega}_{k,C[+1]}\end{pmatrix}$ and $\boldsymbol{\Omega}_F:=\begin{pmatrix}\boldsymbol{\Omega}_1\\ \vdots\\ \boldsymbol{\Omega}_K\end{pmatrix}$.

We define the tensor $\boldsymbol{T}$ such that for a pixel j of the output of the convolutional layer, $\boldsymbol{T}_j$ is a linear application that select the pixels of the input of the convolutional layer that are used to compute the pixel j of the output in a flatten version image (flatten only on the space not on the channels). $\boldsymbol{T}\in\mathbb{R}(H[+1]W[+1],d[+1]d[+1],HW)$ where H and W are the height and width of the intermediate image and $H[+1]$ and $W[+1]$ are the height and width of the output image.

As previously, we have $\boldsymbol{B}^c$ the unfolded version of $\boldsymbol{B}$ such that $\boldsymbol{B}^c\in\mathbb{R}(n,C[-1]dd,HW)$ satifying $\mathbf{Conv}(\boldsymbol{B}_i)$ is equal with the correct reshape to $\boldsymbol{A}\boldsymbol{B}_i^c$.

In addition, we use j as an index on the space of pixel instead of having a couple h,w for height and width. With those notations we have:

$$\boldsymbol{V}(\boldsymbol{A},\boldsymbol{\Omega})[i,m,j]=\mathbf{Conv}_{(C[+1],K)}(\boldsymbol{\Omega})(\mathbf{Conv}_{(K,C[-1])}(\boldsymbol{A})(\boldsymbol{B}_i))[m,j] \tag{70}$$

$$=\sum_k^K\boldsymbol{\Omega}_{m,k}^T\boldsymbol{T}_j\boldsymbol{B}_i^c\boldsymbol{A}_k \tag{71}$$

In the following for simplicity wenote $\boldsymbol{B}_{i,j}^t := \boldsymbol{T}_j \boldsymbol{B}_i^c$. To find the best neurons to add we solve the expressivity bottleneck as :

$$\arg\min_{\boldsymbol{A},\boldsymbol{\Omega}} \frac{1}{n} \sum_i \sum_j \sum_m ||\boldsymbol{V}_{\text{goal}_{\text{proj}_i}}^{(j,m)} - \sum_{k=1}^{K} \boldsymbol{\omega}_{m,k}^T \boldsymbol{B}_{i,j}^t \boldsymbol{\alpha}_k||^2 \tag{72}$$

$$\tag{73}$$

Using the properties of the trace, it follows that :

$$\sum_{i,j,m} ||\boldsymbol{V}_{\text{goal}_{\text{proj}_i}}^{(j,m)} - \sum_{k=1}^{K} \boldsymbol{\omega}_{m,k}^T \boldsymbol{B}_{i,j}^t \boldsymbol{\alpha}_k||^2 = \sum_{i,j,m} ||\boldsymbol{V}_{\text{goal}_{\text{proj}_i}}^{(j,m)} - \sum_k \text{Tr}(\boldsymbol{B}_{i,j}^t \boldsymbol{\alpha}_k \boldsymbol{\omega}_{m,k}^T)||^2 \tag{74}$$

$$= \sum_{i,j,m} ||\boldsymbol{V}_{\text{goal}_{\text{proj}_i}}^{(j,m)} - \text{Tr}(\boldsymbol{B}_{i,j}^t \overbrace{\sum_k \boldsymbol{\alpha}_k \boldsymbol{\omega}_{m,k}^T}^{\boldsymbol{F}_m})||^2 \tag{75}$$

$$= \sum_{i,j,m} ||\boldsymbol{V}_{\text{goal}_{\text{proj}_i}}^{(j,m)} - flat(\boldsymbol{B}_{i,j}^t)^T flat(\boldsymbol{F}_m)||^2 \tag{76}$$

$$= \sum_{i,j} ||\boldsymbol{V}_{\text{goal}_{\text{proj}_i}}^{(j)} - flat(\boldsymbol{B}_{i,j}^t)^T \boldsymbol{F}||^2 \tag{77}$$

$$\tag{78}$$

With $\boldsymbol{F} := \begin{pmatrix} flat(\boldsymbol{F}_1) & \dots & flat(\boldsymbol{F}_{C[+1]}) \end{pmatrix}$.

We remark that $\boldsymbol{V}(\boldsymbol{A}, \boldsymbol{\Omega})$ is a linear function of the matrix $\boldsymbol{F}$ which implies that the solution of 72 is the same as for linear layer. Replacing $\boldsymbol{\Omega A}$ by $\boldsymbol{F}$ in 60 and following the same reasoning as for linear layer, it follows that 72 is equivalent to :

$$\arg\min_{\boldsymbol{F}} \left|\left| \mathbf{S}^{\frac{1}{2}} \boldsymbol{F} - \mathbf{S}^{-\frac{1}{2}} \boldsymbol{N} \right|\right|_2 \tag{79}$$

with $\boldsymbol{S} := \sum_{i,j} flat(\boldsymbol{B}_{i,j}^t) flat(\boldsymbol{B}_{i,j}^t)^T$ and $\boldsymbol{N} := \sum_{i,j} \boldsymbol{V}_{\text{goal}_{\text{proj}}}{}^j flat(\boldsymbol{B}_{i,j}^t)^T$.

However, we remark that the dimension of $\boldsymbol{N} \in \mathbb{R}(C[-1]d[+1]^2 d^2, d[+1])$ is quite large and that computing the SVD of such matrix is costly. To avoid expensive computation, we approximate 72 by defining the matrix $\boldsymbol{S}$ and $\boldsymbol{N}$ as 80 and 82. We now prove that 3.2, 3.4 and equation (132) still hold with such new definitions of $\boldsymbol{S}$ and $\boldsymbol{N}$.

Lemma C.4. *Let $H = \max(\boldsymbol{B}_{i,j}^t.shape)$, we define:*

$$\boldsymbol{S} := \frac{H}{n} \boldsymbol{X} \sum_{i=1}^{n} \sum_{j=1}^{H[+1]W[+1]} (\boldsymbol{B}_{i,j}^t)^\top (\boldsymbol{B}_{i,j}^t) \in (C[-1]dd, C[-1]dd) \tag{80}$$

$$\boldsymbol{N}_m := \frac{1}{n} \sum_{i,j}^{n,H[+1]W[+1]} \boldsymbol{V}_{goal_{proj_{i,j,m}}} (\boldsymbol{B}_{i,j}^t)^\top \in (C[-1]dd, d[+1]d[+1]) \tag{81}$$

$$\boldsymbol{N} := \begin{pmatrix} \boldsymbol{N}_1 \cdots \boldsymbol{N}_{C[+1]} \end{pmatrix} \in (C[-1]dd, C[+1]d[+1]d[+1]) \tag{82}$$

We have:

$$\frac{1}{n} \left|\left| \boldsymbol{V}(\boldsymbol{A}, \boldsymbol{\Omega}) - \boldsymbol{V}_{goal_{proj}} \right|\right|^2 \leq \left|\left| \mathbf{S}^{\frac{1}{2}} \boldsymbol{A}_F \boldsymbol{\Omega}_F - \mathbf{S}^{-\frac{1}{2}} N \right|\right|^2 - \left|\left| \mathbf{S}^{-\frac{1}{2}} \boldsymbol{N} \right|\right|^2 + \frac{1}{n} \left|\left| \boldsymbol{V}_{goal_{proj}} \right|\right|^2 \tag{83}$$

Proof.

$$\frac{1}{n}\left|\left|\boldsymbol{V}(\boldsymbol{A},\boldsymbol{\Omega})-\boldsymbol{V}_{\text{goal}_{proj}}\right|\right|^2 = \frac{1}{n}\sum_{m}^{C[+1]}\sum_{i,j}^{n,H[+1]W[+1]}(\sum_{k}^{K}\boldsymbol{\Omega}_{k,m}^T\boldsymbol{B}_{i,j}^t\boldsymbol{\alpha}_k-\boldsymbol{V}_{\text{goal}_{proj}{}_{i,j,m}})^2 \tag{84}$$

$$(Q :=) \quad = \frac{1}{n}\sum_{m}^{C[+1]}\sum_{i,j}^{n,H[+1]W[+1]}(\sum_{k}^{K}\boldsymbol{\Omega}_{k,m}^T\boldsymbol{B}_{i,j}^t\boldsymbol{\alpha}_k)^2 \tag{85}$$

$$(-2L :=) \quad -\frac{2}{n}\sum_{m}^{C[+1]}\sum_{i,j}^{n,H[+1]W[+1]}\sum_{k}^{K}\boldsymbol{\Omega}_{k,m}^T\boldsymbol{B}_{i,j}^t\boldsymbol{\alpha}_k\boldsymbol{V}_{\text{goal}_{proj}{}_{i,j,m}} \tag{86}$$

$$+\frac{1}{n}\sum_{m}^{C[+1]}\sum_{i,j}^{n,H[+1]W[+1]}\boldsymbol{V}_{\text{goal}_{proj}{}_{i,j,m}}^2 \tag{87}$$

$$= Q-2L+\frac{1}{n}||\boldsymbol{V}_{\text{goal}_{proj}}||^2 \tag{88}$$

We will use the following lemma to simplify this expression.

Lemma C.5. *For any square matrix* $\boldsymbol{A}\in\mathbb{R}^{(n,n)}$,$\text{Tr}(\boldsymbol{A})^2\leq\text{rank}(\boldsymbol{A})\,||\boldsymbol{A}||^2$.

Proof. Using the truncated SVD we have $\boldsymbol{A}=\boldsymbol{U}\Sigma\boldsymbol{V}$ with Σ a diagonal and $\boldsymbol{U}\in\mathbb{R}^{(n,\text{rank}(\boldsymbol{A}))},\boldsymbol{V}\in\mathbb{R}^{(\text{rank}(\boldsymbol{A}),n)}$ truncated orthonormal matrices.

We have:

$$\text{Tr}(\boldsymbol{A})^2 = \text{Tr}(\boldsymbol{U}\Sigma\boldsymbol{V})^2 \tag{89}$$

$$= \text{Tr}(\boldsymbol{V}\boldsymbol{U}\Sigma)^2 \tag{90}$$

$$= \langle\boldsymbol{V}\boldsymbol{U},\Sigma\rangle^2 \tag{91}$$

$$\text{(Cauchy-Swarz)} \quad \leq ||\boldsymbol{V}\boldsymbol{U}||^2\,||\Sigma||^2 \tag{92}$$

As $\boldsymbol{U},\boldsymbol{V}$ are truncated orthonormal matrices, we have:

$$||\boldsymbol{V}\boldsymbol{U}||^2=\text{Tr}(\boldsymbol{U}^T\boldsymbol{V}^T\boldsymbol{V}\boldsymbol{U})=\text{Tr}(\boldsymbol{U}^T\boldsymbol{U})=\text{Tr}(I_{\text{rank}(\boldsymbol{A})})=\text{rank}(\boldsymbol{A})$$

Hence:

$$\text{Tr}(\boldsymbol{A})^2\leq\text{rank}(\boldsymbol{A})\,||\Sigma||^2 \tag{93}$$

As $\boldsymbol{U},\boldsymbol{V}$ are truncated orthonormal matrices, we have:

$$||\Sigma||^2=\text{Tr}(\Sigma^T\Sigma)=\text{Tr}((\boldsymbol{V}\Sigma\boldsymbol{U})\boldsymbol{U}\Sigma\boldsymbol{V})=\text{Tr}(\boldsymbol{A}^T\boldsymbol{A})=||\boldsymbol{A}||^2$$

We conclude that:

$$\text{Tr}(\boldsymbol{A})^2\leq\text{rank}(\boldsymbol{A})\,||\boldsymbol{A}||^2 \tag{94}$$

□

Lemma C.6. *For* $(\boldsymbol{M}_i)_{i\in I}\in(m,n)^I,(\boldsymbol{u}_k)_{k\in[[K]]}\in(m)^K,(\boldsymbol{v}_k)_{k\in[[K]]}\in(n)^K$ *and with* $\boldsymbol{W}:=\sum_{k\in[\![K]\!]}\boldsymbol{v}_k\boldsymbol{u}_k^T\in(n,m)$ *we have:*

$$\sum_{i\in I}\left|\left|\sum_{k\in[\![K]\!]}\boldsymbol{u}_k^T\boldsymbol{M}_i\boldsymbol{v}_k\right|\right|^2\leq\left\langle\boldsymbol{W},\sum_{i\in I}\boldsymbol{M}_i^T\boldsymbol{M}_i\boldsymbol{W}\right\rangle \tag{95}$$

Proof. Let $i \in I$:

$$
\begin{aligned}
||\sum_{k\in[\![K]\!]} \boldsymbol{v}\boldsymbol{u}_k^T \boldsymbol{M}_i \boldsymbol{v}_k||^2 &= \left(\sum_{k\in[\![K]\!]} \boldsymbol{u}_k^T \boldsymbol{M} \boldsymbol{v}_k\right)^2 && (96)\\
&= \mathrm{Tr}(\sum_{k\in[\![K]\!]} \boldsymbol{u}_k^T \boldsymbol{M}_i \boldsymbol{v}_k)^2 && (97)\\
&= \mathrm{Tr}(\boldsymbol{M}_i \sum_{k\in[\![K]\!]} \boldsymbol{v}_k \boldsymbol{u}_k^T)^2 && (98)\\
&= \mathrm{Tr}(\boldsymbol{M}_i \boldsymbol{W})^2 && (99)\\
\text{(theorem C.5)} \quad &\leq \mathrm{rank}(\boldsymbol{M}_i \boldsymbol{W})||\boldsymbol{M}_i \boldsymbol{W}||^2 && (100)\\
&\leq H||\boldsymbol{M}_i \boldsymbol{W}||^2 \quad H := \min(\boldsymbol{M}_i.shape) && (101)
\end{aligned}
$$

Hence we have:

$$
\begin{aligned}
\sum_{i\in I} ||\sum_{k\in[\![K]\!]} \boldsymbol{u}_k^T \boldsymbol{M}_i \boldsymbol{v}_k||^2 &\leq H \sum_{i\in I} ||\boldsymbol{M}_i \boldsymbol{W}||^2 && (102)\\
&= H \sum_{i\in I} \langle \boldsymbol{M}_i \boldsymbol{W}, \boldsymbol{M}_i \boldsymbol{W}\rangle && (103)\\
&= H \sum_{i\in I} \left\langle \boldsymbol{W}, \boldsymbol{M}_i^\top \boldsymbol{M}_i \boldsymbol{W}\right\rangle && (104)\\
&= H \left\langle \boldsymbol{W}, \sum_{i\in I} \boldsymbol{M}_i^\top \boldsymbol{M}_i \boldsymbol{W}\right\rangle && (105)
\end{aligned}
$$

□

Using theorem C.6 we have:

$$
\begin{aligned}
Q &= \frac{1}{n} \sum_{m}^{C[+1]} \sum_{i,j}^{n,H[+1]W[+1]} ||\sum_{k}^{K} \boldsymbol{\Omega}_{k,m}^T (\boldsymbol{B}_{i,j}^t) \boldsymbol{\alpha}_k||^2 && (106)\\
&\leq \sum_{m}^{C[+1]} \left\langle \sum_{k}^{K} \boldsymbol{\alpha}_k \boldsymbol{\Omega}_{k,m}^T, H\frac{1}{n} \sum_{i,j}^{n,H[+1]W[+1]} (\boldsymbol{B}_{i,j}^t)^\top (\boldsymbol{B}_{i,j}^t) \sum_{k}^{K} \boldsymbol{\alpha}_k \boldsymbol{\Omega}_{k,m}^T \right\rangle && (107)\\
&= \sum_{m}^{C[+1]} \langle \boldsymbol{A}_F \boldsymbol{\Omega}_m, \boldsymbol{S}\boldsymbol{A}_F \boldsymbol{\Omega}_m\rangle && (108)\\
&= \langle \boldsymbol{A}_F \boldsymbol{\Omega}_F, \boldsymbol{S}\boldsymbol{A}_F \boldsymbol{\Omega}_F\rangle && (109)
\end{aligned}
$$

Lemma C.7. *For $\boldsymbol{M} \in (m,n), \boldsymbol{u} \in (m), \boldsymbol{v} \in (n)$ we have:*

$$
\boldsymbol{u}^T \boldsymbol{M} \boldsymbol{v} = \left\langle \boldsymbol{v}\boldsymbol{u}^T, \boldsymbol{M}^\top \right\rangle \tag{110}
$$

Proof.

$$
\begin{aligned}
\boldsymbol{u}^T \boldsymbol{M} \boldsymbol{v}_k &= (\boldsymbol{u}^T \boldsymbol{M} v)^\top && (111)\\
&= \boldsymbol{v}^\top \boldsymbol{M}^\top \boldsymbol{u} && (112)\\
&= \left\langle \boldsymbol{v}, \boldsymbol{M}^\top \boldsymbol{u}\right\rangle && (113)\\
&= \left\langle \boldsymbol{v}\boldsymbol{u}^T, \boldsymbol{M}^\top \right\rangle && (114)
\end{aligned}
$$

□

We have:

$$L = \frac{1}{n} \sum_{m}^{C[+1]} \sum_{i,j}^{n,H[+1]W[+1]} \sum_{k}^{K} \boldsymbol{\Omega}_{k,m}^{T} \boldsymbol{B}_{i,j}^{t} \boldsymbol{\alpha}_k \boldsymbol{V}_{\text{goal}_{proj}\,i,j,m} \tag{115}$$

$$(\boldsymbol{V}_{\text{goal}_{proj}\,i,j,m} \in (1)) \quad = \sum_{m}^{C[+1]} \sum_{k}^{K} \boldsymbol{\Omega}_{k,m}^{T} \frac{1}{n} \sum_{i,j}^{n,H[+1]W[+1]} (\boldsymbol{B}_{i,j}^{t} \boldsymbol{V}_{\text{goal}_{proj}\,i,j,m}) \boldsymbol{\alpha}_k \tag{116}$$

$$(\textit{theorem C.7}) \quad = \sum_{m}^{C[+1]} \Big\langle \sum_{k}^{K} \boldsymbol{\alpha}_k \boldsymbol{\Omega}_{k,m}^{T}, \frac{1}{n} \sum_{i,j}^{n,H[+1]W[+1]} \boldsymbol{V}_{\text{goal}_{proj}\,i,j,m} (\boldsymbol{B}_{i,j}^{t})^{\top} \Big\rangle \tag{117}$$

$$= \sum_{m}^{C[+1]} \langle \boldsymbol{A}_F \boldsymbol{\Omega}_m, \boldsymbol{N}_m \rangle \tag{118}$$

$$= \langle \boldsymbol{A}_F \boldsymbol{\Omega}_F, \boldsymbol{N} \rangle \tag{119}$$

In total, we get:

$$\frac{1}{n} \left|\left| \boldsymbol{V}(\boldsymbol{A}, \boldsymbol{\Omega}) - \boldsymbol{V}_{\text{goal}_{proj}} \right|\right|^2 \leq \langle \boldsymbol{A}_F \boldsymbol{\Omega}_F, \boldsymbol{S} \boldsymbol{A}_F \boldsymbol{\Omega}_F \rangle - 2 \langle \boldsymbol{A}_F \boldsymbol{\Omega}_F, \boldsymbol{N} \rangle + \frac{1}{n} ||\boldsymbol{V}_{\text{goal}_{proj}}||^2 \tag{120}$$

If we suppose that $\boldsymbol{S}$ is invertible, we can apply theorem C.1 and get the result.

□

Lemma C.8. *For $\boldsymbol{S} \in \mathbb{R}(s,s), \boldsymbol{N} \in \mathbb{R}(s,t), \boldsymbol{A} \in \mathbb{R}(s,K), \boldsymbol{B} \in \mathbb{R}(K,t)$.*

We note $\boldsymbol{U}\Lambda\boldsymbol{V}$ the singular value decomposition of $\mathbf{S}^{-\frac{1}{2}}\boldsymbol{N}$ and $\boldsymbol{U}_K$ the first K columns of $\boldsymbol{U}$, V_K the first K lines of $\boldsymbol{V}$, Λ_K the first K singular values of Λ and $\Lambda_{K+1:}$ the other singular values of Λ.

We define:

$$\boldsymbol{A}^* := \mathbf{S}^{-\frac{1}{2}} \boldsymbol{U}_K \sqrt{\Lambda_K} \tag{121}$$

$$\boldsymbol{\Omega}^* := \sqrt{\Lambda_K} \boldsymbol{V}_K \tag{122}$$

$$\min_{\boldsymbol{A},\boldsymbol{\Omega}} \frac{1}{n} \left|\left| \boldsymbol{V}(\boldsymbol{A}, \boldsymbol{\Omega}) - \boldsymbol{V}_{goal_{proj}} \right|\right|^2 \leq \frac{1}{n} \left|\left| \boldsymbol{V}(\boldsymbol{A}^*, \boldsymbol{\Omega}^*) - \boldsymbol{V}_{goal_{proj}} \right|\right|^2 = -\,||\Lambda_K||^2 + \frac{1}{n} \left|\left| \boldsymbol{V}_{goal_{proj}} \right|\right|^2 \tag{123}$$

with equality for the linear case.

$$\Psi^{l}_{\theta \oplus \theta^{K\,*}_{\leftrightarrow}} \leq \Psi^{l}_{\theta} - \sum_{k=1}^{K} \lambda_k^2 \tag{124}$$

Proof. Using theorem C.3 and theorem C.4:

$$\frac{1}{n} \left|\left| \boldsymbol{V}(\boldsymbol{A}, \boldsymbol{\Omega}) - \boldsymbol{V}_{\text{goal}_{proj}} \right|\right|^2 \leq \left|\left| \mathbf{S}^{\frac{1}{2}} \boldsymbol{A}\boldsymbol{\Omega} - \mathbf{S}^{-\frac{1}{2}} \boldsymbol{N} \right|\right|^2 - \left|\left| \mathbf{S}^{-\frac{1}{2}} \boldsymbol{N} \right|\right|^2 + \frac{1}{n} \overbrace{\left|\left| \boldsymbol{V}_{\text{goal}_{proj}} \right|\right|^2}^{\Psi^{l}_{\theta}} \tag{125}$$

Hence we have:

$$\arg\min_{\boldsymbol{A},\boldsymbol{\Omega}} \frac{1}{n} \left|\left| \boldsymbol{V}(\boldsymbol{A}, \boldsymbol{\Omega}) - \boldsymbol{V}_{\text{goal}_{proj}} \right|\right| = \arg\min_{\boldsymbol{A},\boldsymbol{\Omega}} \left|\left| \mathbf{S}^{\frac{1}{2}} \boldsymbol{A}\boldsymbol{\Omega} - \mathbf{S}^{-\frac{1}{2}} \boldsymbol{N} \right|\right| \tag{126}$$

As we suppose that S is invertible, we can use the change of variable $\widetilde{\boldsymbol{A}} = \mathbf{S}^{\frac{1}{2}}\boldsymbol{A}$ thus we have:

$$\min_{\boldsymbol{A},\boldsymbol{\Omega}} \left|\left|\mathbf{S}^{\frac{1}{2}}\boldsymbol{A}\boldsymbol{\Omega} - \mathbf{S}^{-\frac{1}{2}}\boldsymbol{N}\right|\right| = \min_{\widetilde{\boldsymbol{A}},\boldsymbol{\Omega}} \left|\left|\widetilde{\boldsymbol{A}}\boldsymbol{\Omega} - \mathbf{S}^{-\frac{1}{2}}\boldsymbol{N}\right|\right| \tag{127}$$

The solution of such problems is given by the paper Eckart & Young (1936) and is:

$$\widetilde{\boldsymbol{A}}^* = \boldsymbol{U}_K\sqrt{\Lambda_K} \tag{128}$$

$$\boldsymbol{\Omega}^* = \sqrt{\Lambda_K}\boldsymbol{V}_K \tag{129}$$

To recover $\boldsymbol{A}^*$ we simply have to multiply by $\mathbf{S}^{-\frac{1}{2}}$ on the left side of $\widetilde{\boldsymbol{A}}^*$. By definition of the SVD and the construction of $(\boldsymbol{A}^*, \boldsymbol{\Omega}^*)$ we have:

$$\left|\left|\mathbf{S}^{\frac{1}{2}}\boldsymbol{A}^*\boldsymbol{\Omega}^* - \mathbf{S}^{-\frac{1}{2}}\boldsymbol{N}\right|\right|^2 - \left|\left|\mathbf{S}^{-\frac{1}{2}}\boldsymbol{N}\right|\right|^2 = ||\Lambda_{K+1:}||^2 - ||\Lambda||^2 = -\,||\Lambda_K||^2 \tag{130}$$

Using this and equation (125) we immediately get the desired equation (123). To conclude, we can also rewrite this with the bottleneck expression:

$$\Psi_{\theta\oplus\theta^K_{\leftrightarrow}} := \min_{\boldsymbol{A},\boldsymbol{\Omega}} \frac{1}{n}\left|\left|\boldsymbol{V}(\boldsymbol{A},\boldsymbol{\Omega}) - \boldsymbol{V}_{\text{goal}_{proj}}\right|\right|^2 \leq \Psi^l_\theta - \sum_{k=1}^{K}\lambda_k^2 \tag{131}$$

□

Proposition C.2. *Solving 8 is equivalent to minimizing the loss $\mathcal{L}$ at order one in $\boldsymbol{V}^l$. Furthermore performing an update of architecture with $\gamma\delta\boldsymbol{W}^*$ (5) and a neuron addition with $\gamma\theta^{K^*}_{\leftrightarrow}$ (3.2), has an impact on the loss at first order in γ as :*

$$\mathcal{L}(f_{\theta\oplus\theta^K_{\leftrightarrow}}) = \mathcal{L}(f_\theta) - \gamma\left(\sigma'_{l-1}(0)\Delta_{\theta^{K,*}_{\leftrightarrow}} + \Delta_{\delta\boldsymbol{W}^*}\right) + o(\gamma) \tag{132}$$

With

$$\Delta_{\theta^{K,*}_{\leftrightarrow}} := \frac{1}{n}\left\langle \boldsymbol{V}^l_{goal_{proj}},\ \boldsymbol{V}^l(\theta^{K,*}_{\leftrightarrow})\right\rangle_{\text{Tr}} = \sum_{k=1}^{K}\lambda_k^2 \tag{133}$$

$$\Delta_{\delta\boldsymbol{W}^*} := \frac{1}{n}\left\langle \boldsymbol{V}^l_{goal},\ \boldsymbol{V}^l(\delta\boldsymbol{W}^*)\right\rangle_{\text{Tr}} \geq 0 \tag{134}$$

To prove note 2 we use the following lemma :

Lemma C.9. *We note $\boldsymbol{V}(\boldsymbol{A},\boldsymbol{\Omega})$ the result of $\boldsymbol{B}$ after applying the layers parameterized by $\boldsymbol{A}$ and $\boldsymbol{\Omega}$. We note $\boldsymbol{V}(\boldsymbol{A}^*,\boldsymbol{\Omega}^*)$ where $\boldsymbol{A}^*$ and $\boldsymbol{B}^*$ as define in 3.2*

$$\frac{1}{n}\left\langle \boldsymbol{V}_{goal_{proj}}, \boldsymbol{V}(\boldsymbol{A}^*,\boldsymbol{\Omega}^*)\right\rangle = ||\Lambda_K||^2 \tag{135}$$

Proof. Starting from theorem C.8

$$\frac{1}{n}\left|\left|\boldsymbol{V}(\boldsymbol{A}^*,\boldsymbol{\Omega}^*) - \boldsymbol{V}_{\text{goal}_{\text{proj}}}\right|\right|^2 = -\,||\Lambda_K||^2 + \frac{1}{n}\left|\left|\boldsymbol{V}_{\text{goal}_{\text{proj}}}\right|\right|^2 \tag{136}$$

Hence by developing the norm, we have:

$$\frac{1}{n}\,||\boldsymbol{V}(\boldsymbol{A}^*,\boldsymbol{\Omega}^*)||^2 - \frac{2}{n}\left\langle \boldsymbol{V}(\boldsymbol{A}^*,\boldsymbol{\Omega}^*), \boldsymbol{V}_{\text{goal}_{\text{proj}}}\right\rangle = -\,||\Lambda_K||^2 \tag{137}$$

Moreover by construction we have $\frac{1}{n}\left|\left|\boldsymbol{V}(\boldsymbol{A}^*,\boldsymbol{\Omega}^*)\right|\right|^2 = \left|\left|\Lambda_K\right|\right|^2$ and therefore we get:

$$-\frac{2}{n}\left\langle \boldsymbol{V}(\boldsymbol{A}^*,\boldsymbol{\Omega}^*), \boldsymbol{V}_{\text{goal}_{\text{proj}}}\right\rangle = -2\left|\left|\Lambda_K\right|\right|^2 \tag{138}$$

which conclude the proof. □

We now prove the note 2. Using the first order approximation of the loss function at $\boldsymbol{a}^l$, we have the following:

$$\mathcal{L}(\boldsymbol{a}^l + \delta\boldsymbol{a}^l) = \mathcal{L}(\boldsymbol{a}^l) + \left\langle \nabla_{\boldsymbol{a}^l}\mathcal{L}, \delta\boldsymbol{a}^l\right\rangle + o(\left|\left|\delta\boldsymbol{a}^l\right|\right|) \tag{139}$$

On one hand, performing an update of architecture, ie $\boldsymbol{W}^* \leftarrow \boldsymbol{W} + \gamma\delta\boldsymbol{W}^*$ change the activation function $\boldsymbol{a}^l$ as $\delta\boldsymbol{a}_1^l := \boldsymbol{V}(\delta\boldsymbol{W}^*)$. Then, as explained in appendix A.5, adding neurons at layer $l-1$ change the activation function $\boldsymbol{a}^l$ as :

$$\delta\boldsymbol{a}_2^l = \sigma'(0)\gamma\boldsymbol{V}(\boldsymbol{A}^*,\boldsymbol{\Omega}^*) + o(\gamma) \tag{140}$$

Combining equation (139) and equation (140), we have :

$$\mathcal{L}(\boldsymbol{A}^*,\boldsymbol{\Omega}^*) = \mathcal{L} + \left\langle \nabla_{\boldsymbol{a}^l}\mathcal{L}, \delta\boldsymbol{a}_1^l + \delta\boldsymbol{a}_2^l\right\rangle + o(\gamma) \tag{141}$$

Using that $\boldsymbol{v}_{\text{goal}}(\boldsymbol{x}_i) := -\nabla_{\boldsymbol{a}^l(\boldsymbol{x}_i)}\mathcal{L}(\boldsymbol{x}_i)$ and that $\mathcal{L} = \frac{1}{n}\sum_i \mathcal{L}(\boldsymbol{x}_i)$, it follows that :

$$\mathcal{L}(\boldsymbol{A}^*,\boldsymbol{\Omega}^*) = \mathcal{L} - \frac{\gamma}{n}\left\langle \boldsymbol{V}_{\text{goal}}, \delta\boldsymbol{a}_1^l + \delta\boldsymbol{a}_2^l\right\rangle + o(\gamma) \tag{142}$$

$$= \mathcal{L} - \frac{\gamma}{n}\left(\left\langle \boldsymbol{V}_{\text{goal}} - \delta\boldsymbol{a}_1^l, \delta\boldsymbol{a}_2^l\right\rangle + \left\langle \delta\boldsymbol{a}_1^l, \delta\boldsymbol{a}_2^l\right\rangle + \left\langle \boldsymbol{V}_{\text{goal}}, \delta\boldsymbol{a}_1^l\right\rangle\right) + o(\gamma) \tag{143}$$

Remarking that $\left\langle \delta\boldsymbol{a}_1^l, \delta\boldsymbol{a}_2^l\right\rangle = o(\gamma)$ and using C.9 we have :

$$\mathcal{L}(\boldsymbol{A}^*,\boldsymbol{\Omega}^*) = \mathcal{L} - \gamma\left(\sigma'(0)\sum_{k=1}^{K}\lambda_k^2 + \frac{1}{n}\left\langle \boldsymbol{V}_{\text{goal}}, \delta\boldsymbol{a}_1^l\right\rangle\right) + o(\gamma) \tag{144}$$

Note on the approximation for convolutional layer. By developing the expression $||\boldsymbol{V} - \boldsymbol{V}_{\text{goal}_{proj}}||^2$, we remark that minimising $||\boldsymbol{V} - \boldsymbol{V}_{\text{goal}_{proj}}||^2$ over $\boldsymbol{V}$ is equivalent to maximising $\langle \boldsymbol{V}, \boldsymbol{V}_{\text{goal}_{proj}}\rangle$ with a constrain on the norm of $\boldsymbol{V}$. This constrain lies in the functional space of the activities and can be reformulated in the parameter space with the matrix $\boldsymbol{S}$ as $||\boldsymbol{A}\boldsymbol{\Omega}^T||_{\boldsymbol{S}} = ||\boldsymbol{V}||$. By changing the matrix $\boldsymbol{S}$, we modify the metric on $\boldsymbol{V}$ and obtain a pseudo solution $\boldsymbol{S}_{pseudo}^{-1}\boldsymbol{N}$ which is still positively correlated with $\boldsymbol{S}^{-1}\boldsymbol{N}$ as $\langle \boldsymbol{S}^{-1}\boldsymbol{N}, \boldsymbol{S}_{pseudo}^{-1}\boldsymbol{N}\rangle \geq 0$.

C.3 Proposition and remark 3.4 and 3.2

Proposition C.3. *3.4 Suppose $\boldsymbol{S}$ is semi definite, we note $\boldsymbol{S} = \boldsymbol{S}^{\frac{1}{2}}\boldsymbol{S}^{\frac{1}{2}}$. Solving (7) is equivalent to find the K first eigenvectors $\boldsymbol{\alpha}_k$ associated to the K largest eigenvalues λ of the generalized eigenvalue problem :*

$$\boldsymbol{N}\boldsymbol{N}^T\boldsymbol{\alpha}_k = \lambda\boldsymbol{S}\boldsymbol{\alpha}_k \tag{145}$$

Corollary 2. *(3.2)For all integers m, m' such that $m + m' \leqslant R$, at order one in η, adding $m + m'$ neurons simultaneously according to the previous method is equivalent to adding m neurons then m' neurons by applying successively the previous method twice.*

Proof

To prove 3.4, we show that the solution of 145 and the formula of 3.2 are collinear.

Solving 145 is equivalent to maximizing the following generalized Rayleigh quotient (which is solvable by the LOBPCG technique):

$$\boldsymbol{\alpha}^* = \arg\max_{\alpha} \frac{\boldsymbol{\alpha}^T \boldsymbol{N}\boldsymbol{N}^T \boldsymbol{\alpha}}{\boldsymbol{\alpha}^T \boldsymbol{S} \boldsymbol{\alpha}} \tag{146}$$

$$\boldsymbol{p}^* = \arg\max_{\boldsymbol{p}=\boldsymbol{S}^{1/2}\boldsymbol{\alpha}} \frac{\boldsymbol{p}^T \boldsymbol{S}^{-\frac{1}{2}} \boldsymbol{N}\boldsymbol{N}^T \boldsymbol{S}^{-\frac{1}{2}} \boldsymbol{p}}{\boldsymbol{p}^T \boldsymbol{p}} \tag{147}$$

$$\boldsymbol{p}^* = \arg\max_{||\boldsymbol{p}||=1} ||\boldsymbol{N}^T \boldsymbol{S}^{-\frac{1}{2}} \boldsymbol{p}|| \tag{148}$$

$$\boldsymbol{\alpha}^* = \boldsymbol{S}^{-\frac{1}{2}} \boldsymbol{p}^* \tag{149}$$

Considering the SVD of $\boldsymbol{N}^T \boldsymbol{S}^{-\frac{1}{2}} = \sum_{r=1}^{R} \lambda_r \boldsymbol{e}_r \boldsymbol{f}_r^T$, then $\boldsymbol{S}^{-\frac{1}{2}} \boldsymbol{N}\boldsymbol{N}^T \boldsymbol{S}^{-\frac{1}{2}} = \sum_{r=1}^{R} \lambda_r^2 \boldsymbol{f}_r \boldsymbol{f}_r^T$, because $j \neq i \implies \boldsymbol{e}_i^T \boldsymbol{e}_j = 0$ and $\boldsymbol{f}_i^T \boldsymbol{f}_j = 0$. Hence maximizing the first quantity is equivalent to $\boldsymbol{p}_k^* = \boldsymbol{f}_k$, then $\boldsymbol{\alpha}_k = \boldsymbol{S}^{-\frac{1}{2}} \boldsymbol{f}_k$, which match the formula of proposition 3.2. The same reasoning can be applied on $\boldsymbol{\omega}_k$.

We prove second corollary 3.2 by induction. Note that $\boldsymbol{v}(\theta_{\leftrightarrow}^{K,*}, \boldsymbol{x}) = o(\eta)$, then for $m = m' = 1$:

$$\boldsymbol{a}_l(\boldsymbol{x})^{t+1} = \boldsymbol{a}_l(\boldsymbol{x})^t + \boldsymbol{v}(\theta_{\leftrightarrow}^{1,*}, \boldsymbol{x}) + o(\eta) \tag{150}$$

Remark that $\boldsymbol{v}_{\text{goal}}(\boldsymbol{x})$ is a function of $\boldsymbol{a}_l(\boldsymbol{x})$, ie $\boldsymbol{v}_{\text{goal}}(\boldsymbol{x}) := g(\boldsymbol{a}_l(\boldsymbol{x}))$. Then suppose that $\mathcal{L}(f(\boldsymbol{x}), \boldsymbol{y}))$ is twice differentiable in $\boldsymbol{a}_l(\boldsymbol{x})$. It follows that $g(\boldsymbol{a}_l(\boldsymbol{x}))$ is differentiable and :

$$\boldsymbol{v}_{\text{goal}}{}^{t+1}(\boldsymbol{x}) = g(\boldsymbol{a}_l^t(\boldsymbol{x}) + \boldsymbol{v}(\theta_{\leftrightarrow}^{1,*}, \boldsymbol{x})) \tag{151}$$

$$= g(\boldsymbol{a}_l^t(\boldsymbol{x})) + \nabla_{\boldsymbol{a}_l^t(\boldsymbol{x})} g(\boldsymbol{a}_l^t(\boldsymbol{x}))^T \boldsymbol{v}(\theta_{\leftrightarrow}^{1,*}, \boldsymbol{x}) + o(\eta^2) \tag{152}$$

$$= \boldsymbol{v}_{\text{goal}}{}^t(\boldsymbol{x}) + \eta \frac{\partial^2 \mathcal{L}(f_\theta(\boldsymbol{x}), \boldsymbol{y})}{\partial \boldsymbol{a}^l(\boldsymbol{x})^2} \boldsymbol{v}(\theta_{\leftrightarrow}^{1,*}, \boldsymbol{x}) + o(\eta^2) \tag{153}$$

$$= \boldsymbol{v}_{\text{goal}}{}^t(\boldsymbol{x}) + o(\eta) \tag{154}$$

Adding the second neuron we obtain the minimization problem:

$$\arg\min_{\boldsymbol{\alpha}_2, \boldsymbol{\omega}_2} ||\boldsymbol{V}_{\text{goal}}{}^t - \boldsymbol{V}(\boldsymbol{\alpha}_2, \boldsymbol{\omega}_2)|| + o(\eta) \tag{155}$$

□

C.4 About equivalence of quadratic problems

Problems 8 and 7 are generally not equivalent, but might be very close, depending on layer sizes and number of samples. The difference between the two problems is that in one case one minimizes the quadratic quantity:

$$\left\| \boldsymbol{V}^l(\theta_{\leftrightarrow}^K) + \boldsymbol{V}^l(\boldsymbol{M}) - \boldsymbol{V}_{\text{goal}}{}^l \right\|^2$$

w.r.t. $\boldsymbol{M}$ and $\theta_{\leftrightarrow}^K$ **jointly**, while in the other case the problem is first minimized w.r.t. $\boldsymbol{M}$ and then w.r.t. $\theta_{\leftrightarrow}^K$. The latter process, being greedy, might thus provide a solution that is not as optimal as the joint optimization.

We chose this two-step process as it intuitively relates to the spirit of improving upon a standard gradient descent: we aim at adding neurons that complement what the other ones have already done. This choice is debatable and one could solve the joint problem instead, with the same techniques.

The topic of this section is to check how close the two problems are. To study this further, note that $\boldsymbol{V}^l(\boldsymbol{M}) = \delta \boldsymbol{W}_l \boldsymbol{B}_{l-1}$ while $\boldsymbol{V}^l(\theta_{\leftrightarrow}^K) = \sum_{k=1}^{K} \boldsymbol{\omega}_k \boldsymbol{B}_{l-2}^T \boldsymbol{\alpha}_k$. The rank of $\boldsymbol{B}_{l-1}$ is $\min(n_S, n_{l-1})$ where n_S is the number of samples and n_{l-1} the number of neurons (post-activities) in layer $l-1$, while the rank of $\boldsymbol{B}_{l-2}$ is $\min(n_S, n_{l-2})$ where n_{l-2} is the number of neurons (post-activities) in layer $l-2$. Note also that the number of degrees of freedom in the optimization variables $\delta \boldsymbol{W}_l$ and $\theta_{\leftrightarrow}^K = (\boldsymbol{\omega}_k, \boldsymbol{\alpha}_k)$ is much larger than these ranks.

Small sample case. If the number n_S of samples is lower than the number of neurons n_{l-1} and n_{l-2} (which is potentially problematic, see Section E.1), then it is possible to find suitable variables $\delta \boldsymbol{W}_l$ and $\theta^K_{\leftrightarrow}$ to form any desired $\boldsymbol{V}^l(\boldsymbol{M})$ and $\boldsymbol{V}^l(\theta^K_{\leftrightarrow})$. In particular, if $n_S \leqslant n_{l-1} \leqslant n_{l-2}$, one can choose $\boldsymbol{V}^l(\theta^K_{\leftrightarrow})$ to be ${\boldsymbol{V}_{\text{goal}}}^l - \boldsymbol{V}^l(\boldsymbol{M})$ and thus cancel any effect due to the greedy process in two steps. The two problems are then equivalent.

Large sample case. On the opposite, if the number of samples is very large (compared to the number of neurons n_{l-1} and n_{l-2}), then the lines of matrices $\boldsymbol{B}_{l-1}$ and $\boldsymbol{B}_{l-2}$ become asymptotically uncorrelated, under the assumption of their independence (which is debatable, depending on the type of layers and activation functions). Thus the optimization directions available to $\boldsymbol{V}^l(\boldsymbol{M})$ and $\boldsymbol{V}^l(\theta^K_{\leftrightarrow})$ become orthogonal, and proceeding greedily does not affect the result, the two problems are asymptotically equivalent.

In the general case, matrices $\boldsymbol{B}_{l-1}$ and $\boldsymbol{B}_{l-2}$ are not independent, though not fully correlated, and the number of samples (in the minibatch) is typically larger than the number of neurons; the problems are then different.

Note that technically the ranks could be lower, in the improbable case where some neurons are perfectly redundant, or, e.g., if some samples yield exactly the same activities.

D Section *About greedy growth sufficiency and TINY convergence* with more details and proofs

One might wonder whether a greedy approach on layer growth might get stuck in a non-optimal state. By *greedy* we mean that every neuron added has to decrease the loss. We derive the following series of propositions in this regard. Since in this work we add neurons layer per layer independently, we study here the case of a single hidden layer network, to spot potential layer growth issues. For the sake of simplicity, we consider the task of least square regression towards an explicit continuous target f^*, defined on a compact set. That is, we aim at minimizing the loss:

$$\inf \sum_{\boldsymbol{x} \in \mathcal{D}} \|f(\boldsymbol{x}) - f^*(\boldsymbol{x})\|^2 \tag{156}$$

where $f(\boldsymbol{x})$ is the output of the neural network and $\mathcal{D}$ is the training set.

We start with an optional introductive section D.1 about greedy growth possibilities, then prepare lemmas in Sections D.2 and D.3 that will be used in Section D.4 to show that one can keep on adding neurons to a network (without modifying already existing weights) to make it converge exponentially fast towards the optimal function. Then in Section D.6 we present a growth method that explicitly overfits each dataset sample one by one, thus requiring only n neurons, thanks to existing weights modification. Finally, more importantly, in Section D.7, we show that actually any reasonable growth method that follows a certain optimization protocol (this includes TINY completed by random neuron additions if necessary) will reach 0 training error in at most n neuron additions.

D.1 Possibility of greedy growth

Proposition D.1 (Greedy completion of an existing network)**.** *If f is not f^* yet, there exists a set of neurons to add to the hidden layer such that the new function f' will have a lower loss than f.*

One can even choose the added neurons such that the loss is arbitrarily well minimized.

Proof. The classic universal approximation theorem about neural networks with one hidden layer Pinkus (1999) states that for any continuous function g^* defined on a compact set $\boldsymbol{\omega}$, for any desired precision γ, and for any activation function σ provided it is not a polynomial, then there exists a neural network g with one hidden layer (possibly quite large when γ is small) and with this activation function σ, such that

$$(.*?) \tag{157}$$

We apply this theorem to the case where $g^* = f^* - f$, which is continuous as f^* is continuous, and f is a shallow neural network and as such is a composition of linear functions and of the function σ, that we will suppose to be continuous for the sake of simplicity. We will suppose that f is real-valued for the sake of simplicity as well, but the result is trivially extendable to vector-valued functions (just concatenate the networks obtained for each output independently). We choose $\gamma = \frac{1}{10}\|f^* - f\|_{L2}$, where $\langle a|\, b\rangle_{L2} = \frac{1}{|\boldsymbol{\omega}|}\int_{\boldsymbol{x}\in\boldsymbol{\omega}} a(\boldsymbol{x})\, b(\boldsymbol{x})\, d\boldsymbol{x}$. This way we obtain a one-hidden-layer neural network g with activation function σ such that:

$$\forall \boldsymbol{x} \in \boldsymbol{\omega},\ \ -\gamma \leqslant g(\boldsymbol{x}) - g^*(\boldsymbol{x}) \leqslant \gamma \tag{158}$$

$$\forall \boldsymbol{x} \in \boldsymbol{\omega},\ \ g(\boldsymbol{x}) = f^*(\boldsymbol{x}) - f(\boldsymbol{x}) + a(\boldsymbol{x}) \tag{159}$$

with $\forall \boldsymbol{x} \in \boldsymbol{\omega},\ |a(\boldsymbol{x})| \leqslant \gamma$.

Then:

$$\forall \boldsymbol{x} \in \boldsymbol{\omega},\ \ f^*(\boldsymbol{x}) - (f(\boldsymbol{x}) + g(\boldsymbol{x})) = -a(\boldsymbol{x}) \tag{160}$$

$$\forall \boldsymbol{x} \in \boldsymbol{\omega},\ \ (f^*(\boldsymbol{x}) - h(\boldsymbol{x}))^2 = a^2(\boldsymbol{x}) \tag{161}$$

with h being the function corresponding to a neural network consisting in concatenating the hidden layer neurons of f and g, and consequently summing their outputs.

$$\|f^* - h\|_{L2}^2 = \|a\|_{L2}^2 \tag{162}$$

$$\|f^* - h\|_{L2}^2 \leqslant \gamma^2 = \frac{1}{100}\|f^* - f\|_{L2}^2 \tag{163}$$

and consequently the loss is reduced indeed (by a factor of 100 in this construction).

The same holds in expectation or sum over a training set, by choosing $\gamma = \frac{1}{10}\sqrt{\frac{1}{|\mathcal{D}|}\sum_{\boldsymbol{x}\in\mathcal{D}} \|f(\boldsymbol{x}) - f^*(\boldsymbol{x})\|^2}$, as Equation (161) then yields:

$$\sum_{\boldsymbol{x}\in\mathcal{D}} (f^*(\boldsymbol{x}) - h(\boldsymbol{x}))^2 = \sum_{\boldsymbol{x}\in\mathcal{D}} a^2(\boldsymbol{x}) \leqslant \frac{1}{100}\sum_{\boldsymbol{x}\in\mathcal{D}} (f^*(\boldsymbol{x}) - f(\boldsymbol{x}))^2 \tag{164}$$

which proves the proposition as stated.

For more general losses, one can consider order-1 (linear) developpment of the loss and ask for a network g that is close to (the opposite of) the gradient of the loss.

□

Proof of the additional remark. The proof in Pinkus (1999) relies on the existence of real values c_n such that the n-th order derivatives $\sigma^{(n)}(c_n)$ are not 0. Then, by considering appropriate values arbitrarily close to c_n, one can approximate the n-th derivative of σ at c_n and consequently the polynomial c^n of order n. This standard proof then concludes by density of polynomials in continuous functions.

Provided the activation function σ is not a polynomial, these values c_n can actually be chosen arbitrarily, in particular arbitrarily close to 0. This corresponds to choosing neuron input weights arbitrarily close to 0. □

Proposition D.2 (Greedy completion by one single neuron)**.** *If f is not f^* yet, there exists a neuron to add to the hidden layer such that the new function f' will have a lower loss than f.*

Proof. From the previous proposition, there exists a finite set of neurons to add such that the loss will be decreased. In this particular setting of $L2$ regression, or for more general losses if considering small function moves, this means that the function represented by this set of neurons has a strictly negative component over the gradient g of the loss ($g = -2(f^* - f)$ in the case of the $L2$ regression). That is, denoting by $a_i\sigma(\boldsymbol{W}_i \cdot \boldsymbol{x})$ these N neurons:

$$\big\langle \sum_{i=1}^{N} a_i\sigma(\boldsymbol{w}_i \cdot \boldsymbol{x}) \,\big|\, g\big\rangle_{L2} = K < 0 \tag{165}$$

i.e.

$$\sum_{i=1}^{N} \langle a_i \sigma(\boldsymbol{w}_i \cdot \boldsymbol{x}) |\, g \rangle_{L2} = K < 0 \tag{166}$$

We have:

$$0 > \frac{1}{N} K = \frac{1}{N} \sum_{i=1}^{N} \langle a_i \sigma(\boldsymbol{w}_i \cdot \boldsymbol{x}) |\, g \rangle_{L2} \geq \min_{i=1}^{N} \langle a_i \sigma(\boldsymbol{w}_i \cdot \boldsymbol{x}) |\, g \rangle_{L2} \tag{167}$$

Then necessarily at least one of the N neurons satisfies

$$\langle a_i \sigma(\boldsymbol{w}_i \cdot \boldsymbol{x}) |\, g \rangle_{L2} \leqslant \frac{1}{N} K < 0 \tag{168}$$

and thus decreases the loss when added to the hidden layer of the neural network representing f. Moreover this decrease is at least $\frac{1}{N}$ of the loss decrease resulting from the addition of all neurons.

□

As a consequence, there exists no situation where one would need to add many neurons simultaneously to decrease the loss: it is always feasible with a single neuron. Note that finding the optimal neuron to add is actually NP-hard (Bach, 2017), so we will not necessarily search for the optimal one. A constructive lower bound on how much the loss can be improved will be given later in this section.

Proposition D.3 (Greedy completion by one infinitesimal neuron)**.** *The neuron in the previous proposition can be chosen to have arbitrarily small input weights.*

Proof. This is straightforward, as, following a previous remark, the neurons found to collectively decrease the loss can be supposed to all have arbitrarily small input weights. □

This detail is important in that our approach is based on the tangent space of the function f and thus manipulates infinitesimal quantities. Our optimization problem indeed relies on the linearization of the activation function by requiring the added neuron to have infinitely small input weights, to make the problem easier to solve. This proposition confirms that such neuron exists indeed.

Correlations and higher orders. Note that, as a matter of fact, our approach exploits linear correlations between inputs of a layer and desired output variations. It might happen that the loss is not minimized yet but there is no such correlation to exploit anymore. In that case the optimization problem (8) will not find neurons to add. Yet following Prop. D.3 there does exist a neuron with arbitrarily small input weights that can reduce the loss. This paradox can be explained by pushing further the Taylor expansion of that neuron output in terms of weight amplitude (single factor ε on all of its input weights), for instance $\sigma(\varepsilon \boldsymbol{\alpha} \cdot \boldsymbol{x}) \simeq \sigma(0) + \sigma'(0)\varepsilon \boldsymbol{\alpha} \cdot \boldsymbol{x} + \frac{1}{2}\sigma''(0)\varepsilon^2 (\boldsymbol{\alpha} \cdot \boldsymbol{x})^2 + O(\varepsilon^3)$. Though the linear term $\boldsymbol{\alpha} \cdot \boldsymbol{x}$ might be uncorrelated over the dataset with desired output variation $v(\boldsymbol{x})$, i.e. $\mathbb{E}_{\boldsymbol{x} \sim \mathcal{D}}[\boldsymbol{x}\, v(\boldsymbol{x})] = 0$, the quadratic term $(\boldsymbol{\alpha} \cdot \boldsymbol{x})^2$, or higher-order ones otherwise, might be correlated with $v(\boldsymbol{x})$. Finding neurons with such higher-order correlations can be done by increasing accordingly the power of $(\boldsymbol{\alpha} \cdot \boldsymbol{x})$ in the optimization problem (7). Note that one could consider other function bases than the polynomials from Taylor expansion, such as Hermite or Legendre polynomials, for their orthogonality properties. In all cases, one does not need to solve such problems exactly but just to find an approximate solution, i.e. a neuron improving the loss.

Adding random neurons. Another possibility to suggest additional neurons, when expressivity bottlenecks are detected but no correlation (up to order p) can be exploited anymore, is to add random neurons. The first p order Taylor expansions will show 0 correlation with desired output variation, hence no loss improvement nor worsening, but the correlation of the $p+1$-th order will be non-0, with probability 1, in the spirit of random projections. Furthermore, in the spirit of common neural network training practice, one could consider brute force combinatorics by adding many random neurons and hoping some will be close enough to the desired direction (Frankle & Carbin, 2018). The difference with usual training is that we would perform such computationally-costly searches only when and where relevant, exploiting all simple information first (linear correlations in each layer).

D.2 Loss decrease with a line search on a quadratic energy

Let $\mathcal{L}$ be a quadratic loss over $\mathbb{R}^d$ and g be a vector in $\mathbb{R}^d$. The loss $\mathcal{L}$ can be written as:

$$\mathcal{L}(g) = g^T Q g + v^T g + K \tag{169}$$

where Q is a matrix that we will suppose to be symmetric positive definite. This is to ensure that all eigenvalues of Q are positive, hence modelling a local minimum without saddle point. v is a vector in $\mathbb{R}^d$ and K is a real constant.

For instance, the mean square loss $\mathbb{E}_{x\in\mathcal{D}}\left[\|f(x)-f^*(x)\|_S^2\right]$, where $\mathcal{D}$ is a finite dataset of N samples, f^* a target function, and S is a symmetric positive definite matrix used as a metric, fits these hypotheses, considering $g = (f(x_1), f(x_2), ...)$ as a vector. Indeed this loss rewrites as

$$\sum_{i=1}^{N} f(x_i)^T S f(x_i) - 2\sum_i f^{*T}(x_i) S f(x_i) + K \;=\; g^T Q\, g \;+\; v^T g \;+\; K \tag{170}$$

by flattening and concatenating the vectors $f(x_i)$ and considering $Q = S \otimes S \otimes S \otimes ...$ the tensor product of N times the same matrix S, i.e. a diagonal-block matrix with N identical blocks S. Note that for the standard regression with the L^2 metric, this matrix Q is just the Identity.

Starting from point g, and given a direction $h \in \mathbb{R}^d$, the question is to perform a line search in that direction, i.e. to optimize the factor $\lambda \in \mathbb{R}$ in order to minimise $\mathcal{L}(g + \lambda h)$.

Developing that expression, we get:

$$\mathcal{L}(g+\lambda h) \;=\; (g+\lambda h)^T Q\,(g+\lambda h) + v^T(g+\lambda h) + K \;=\; \lambda^2 h^T Q h + \lambda(2h^T Q g + v^T h) + \mathcal{L}(g) \tag{171}$$

which is a second-order polynomial in λ with positive quadratic coefficient. Note that the linear coefficient is $h^T \nabla_g \mathcal{L}(g)$, where $\nabla_g \mathcal{L}(g) = 2Qg + v$ is the gradient of $\mathcal{L}$ at point g. The unique minimum of the polynomial in λ is then:

$$\lambda^* = -\frac{1}{2}\frac{h^T \nabla_g \mathcal{L}(g)}{h^T Q h} \tag{172}$$

which leads to

$$\min_\lambda \mathcal{L}(g+\lambda h) = \lambda^{*2} h^T Q h + \lambda^* h^T \nabla_g \mathcal{L}(g) + \mathcal{L}(g) \tag{173}$$

$$= \mathcal{L}(g) - \frac{1}{4}\frac{\left(h^T \nabla_g \mathcal{L}(g)\right)^2}{h^T Q h} \tag{174}$$

$$= \mathcal{L}(g) - \frac{1}{4}\left\langle \frac{h}{\|h\|_Q} \middle| \nabla_g^Q \mathcal{L}(g) \right\rangle_Q^2 . \tag{175}$$

Thus the loss gain obtained by a line search in a direction h is quadratic in the angle between that direction and the gradient of the loss, in the sense of the Q norm (and it is also quadratic in the norm of the gradient). Note that inner products with the gradient do not depend on the metric, in the sense that $\langle h \,|\, \nabla_g \mathcal{L}(g) \rangle_{L^2} \;=\; \langle h \,|\, \nabla_g^S \mathcal{L}(g) \rangle_S \quad \forall h$ for any metric S, i.e. any symmetric definite positive matrix S, associated to the norm $\|h\|_S^2 = h^T S h$ and to the gradient $\nabla_g^S \mathcal{L}(g) = S^{-1} \nabla_g^{L^2} \mathcal{L}(g)$.

In the case of a standard L^2 regression this boils down to:

$$\min_\lambda \|g + \lambda h\|_{L2}^2 = \|g\|^2 - \left\langle \frac{h}{\|h\|} \middle| g \right\rangle_{L2}^2 \tag{176}$$

i.e. considering $\mathcal{L}(f) := \mathbb{E}_{x\in\mathcal{D}}\left[\|f(x)-f^*(x)\|^2\right]$:

$$\min_\lambda \mathcal{L}(f+\lambda h) \;=\; \mathcal{L}(f) - \left\langle \frac{h}{\|h\|} \middle| f^* - f \right\rangle_{L2}^2 \;=\; \mathcal{L}(f) - \frac{\mathop{\mathbb{E}}_{x\in\mathcal{D}}\left[\,(f^*-f)\,h\,\right]^2}{\mathop{\mathbb{E}}_{x\in\mathcal{D}}\left[\,\|h\|^2\,\right]} . \tag{177}$$

D.3 Expected loss gain with a line search in a random direction

Using Appendix D.2 above, the loss gain when performing a line search on a quadratic loss is quadratic in the angle $\alpha = \left\langle \frac{\boldsymbol{V}(X)}{\|\boldsymbol{V}(X)\|} \,\middle|\, \frac{\boldsymbol{V}_{\text{goal}}(X)}{\|\boldsymbol{V}_{\text{goal}}(X)\|} \right\rangle_{L2}$ between the random search direction $\boldsymbol{V}(X)$ and the gradient $\boldsymbol{V}_{\text{goal}}(X)$.

This angle has average 0 and is of standard deviation $\frac{1}{nd}$, as described in Section 5.2. The loss gain is thus of the order of magnitude of $\frac{1}{d}$ in the best case (single-sample minibatch).

D.4 Exponential convergence to 0 training error

Considering a regression to a target f^* with the quadratic loss, the function f represented by the current neural network (fully-connected, one hidden layer, with ReLU activation function) can be improved to reach 0 loss by an addition of n neurons $(h_i)_{1 \leqslant i \leqslant n}$, with n is the dataset size, using Zhang et al. (2017). Unfortunately there is no guarantee that if one adds each of these neurons one by one, the loss decreases each time. We will prove that one of these neurons does decrease the loss, and we will quantify by how much, relying on the explicit construction in Zhang et al. (2017). This decrease will actually be a constant factor of the loss, thus leading to exponential convergence towards the target f^* on the training set.

As in the proof of Proposition D.2 in Appendix D, at least one of the added neurons satisfies that its inner product with the gradient direction is at least $1/n$. While one could consequently hope for a loss gain in $O(\frac{1}{n})$, one has to see that this decrease would be the one of a gradient step, which is multiplied by a step size η, and asks for multiple steps to be done. Instead in our approach we actually perform a line search over the direction of the new neuron. In both cases (line search or multiple small gradient steps), one has to take into account at least order-2 changes of the loss to compute the line search or estimate suitable η and/or its associated number of steps. Luckily in our case of least square regression, the loss is exactly equal to its second order Taylor development, and all following computations are exact.

We consider the mean square regression loss $\mathcal{L}(f) = \mathbb{E}_{x \in \mathcal{D}} \left[\|f(x) - f^*(x)\|_S^2 \right]$, where $\mathcal{D}$ is a finite training dataset of N samples. Its functional gradient $\nabla \mathcal{L}(f)$ at point f is $2(f - f^*)$, which is proportional to the optimal change to add to f, that is, $f^* - f$. The n neurons $(h_i)_{1 \leqslant i \leqslant n}$ to be added to f following Zhang et al. (2017) satisfy $\sum_i h_i = f^* - f = -\frac{1}{2} \nabla \mathcal{L}(f)$. Thus

$$\left\langle \sum_i h_i \,\middle|\, f^* - f \right\rangle_{L2} \;=\; \|f^* - f\|_{L2}^2 \;=\; \mathcal{L}(f)\,. \tag{178}$$

Then like in the proof of D.2 we use that the maximum is greater or equal to the mean to get that there exists a neuron h_i that satisfies:

$$\langle h_i |\; f^* - f \rangle_{L2} \;\geqslant\; \mathcal{L}(f)/n\,. \tag{179}$$

By applying Appendix D.2 one obtains that the new loss after line search into the direction of h_i yields:

$$\min_{\lambda} \mathcal{L}(f + \lambda h_i) \;=\; \mathcal{L}(f) \;-\; \frac{\langle h_i \mid f^* - f \rangle_{L2}^2}{\|h_i\|^2} \;\leqslant\; \mathcal{L}(f) \times \left(1 - \frac{\mathcal{L}(f)}{n^2 \|h_i\|^2} \right)\,. \tag{180}$$

From the particular construction in Zhang et al. (2017) it is possible to bound the square norm of the neuron $\|h_i\|^2$ by $n\, d' \left(\frac{d_M}{d_m}\right)^2 \mathcal{L}(f)$, where d_M is related to the maximum distance between 2 points in the dataset, d_m is another geometric quantity related to the minimum distance, and d' is the network output dimension. To ease the reading of this proof, we defer the construction of this bound to next section, Appendix D.5.

Then the loss at each neuron addition decreases by a factor which is at least $\gamma = 1 - \frac{1}{n^3 d'} \left(\frac{d_m}{d_M}\right)^2 < 1$. This factor is a constant, as it is a bound that depends only on the geometry of the dataset (not on f).

Thus it is possible to decrease the loss exponentially fast with the number t of added neurons, i.e. $\mathcal{L}(f_t) \leqslant \gamma^t \mathcal{L}(f)$, towards 0 training loss, and this in a greedy way, that is, by adding neuron one by one, with the property that each neuron addition decreases the loss.

Note that, in the proof of Zhang et al. (2017), the added neurons could be chosen to have arbitrarily small input weights. This corresponds to choosing a with small norm instead of unit norm in Equation 181.

The number of neuron additions expected to reach good performance according to this bound is in the order of magnitude of n^3, which is to be compared to n (number of neurons needed to overfit the dataset, without the constraint that each addition decreases the loss). This bound might be improved using other constructions than Zhang et al. (2017), though with this proof the bound cannot be better than n^2 (supposing $\|h_i\|$ can be made not to depend on n).

Note also that with ReLU activation functions, all points that are on the convex hull of the dataset (which is necessarily the case of all points if the input dimension is higher that the number of points) can easily in turn be perfectly predicted (0 loss) by just one neuron addition each (without changing the outputs for the other points), by choosing an hyperplane that separates the current convex hull point from the rest of the dataset, and setting a ReLU neuron in that direction.

D.5 Bound on the norm of the neurons

Here we prove that the neurons obtained by Zhang et al. (2017) can be chosen so as to bound the square norm of any neuron $\|h_i\|^2$ by $n\, d' \left(\frac{d_M}{d_m}\right)^2 \mathcal{L}(f)$, where d_M is related to the maximum distance between 2 points in the dataset, and d_m is another geometric quantity related to the minimum distance. For the sake of simplicity, we first consider the case where the output dimension is $d' = 1$.

In Zhang et al. (2017), the n neurons are obtained by solving $y = Aw$, where $y = (y_1, y_2 ...)$ is the target function (here $(f^* - f)$ at each x_j), A is the matrix given by $A_{jk} = \mathrm{ReLU}(a \cdot x_j - b_k)$, representing neuron activations, and a is any vector that separates the dataset points, i.e. $a \cdot x_j \neq a \cdot x_{j'}\ \forall j \neq j'$, that is, a could be almost any vector in $\mathbb{R}^d$ (in the sense of random projections, that is, the set of vectors that do not satisfy this is of measure 0).

Here we will pick a particular unit direction a, one that maximizes the distance between any two samples after projection:

$$a \in \underset{\|a\|=1}{\arg\max}\ \min_{j,j'} |a \cdot (x_j - x_{j'})| \tag{181}$$

and let us denote d'_m the associated value: $d'_m = \min_{j,j'} |a \cdot (x_j - x_{j'})|$ for that a. Note that $d'_m \leqslant \min_{j,j'} \|x_j - x_{j'}\|$ and that it depends only on the training set. The quantity d'_m is likely to be also lower-bounded (over all possible datasets) by $\min_{j,j'} \|x_j - x_{j'}\|$ times a factor depending on the embedding dimension d and the number of points n.

Now, let us sort the samples according to increasing $a \cdot x_j$, that is, let us re-index the samples such that $(a \cdot x_j)$ now grows with j. By definition of a, the difference between any two consecutive $a \cdot x_j$ is at least d'_m.

We now choose biases $b_j = a \cdot x_j - d'_m + \varepsilon$ for some very small ε. The neurons are then defined as $h_k(x) = w_k \mathrm{ReLU}(a \cdot x - b_k)$. The induced activation matrix $A_{jk} = \mathrm{ReLU}(a \cdot x_j - b_k)$ then satisfies $\forall j < k; A_{jk} = 0$ and $\forall j \geqslant k; A_{jk} \geqslant d'_m - \varepsilon$. The matrix A is lower triangular with diagonal elements above $d_m := d'_m - \varepsilon$, hence invertible. Recall that $y = Aw$.

Consequently, $w = A^{-1}y$, and hence $\|w\|^2 \leqslant |||A^{-1}|||^2 \|y\|^2$, that is,

$$\|w\|^2 \leqslant \frac{1}{d_m^2} \mathcal{L}(f) \tag{182}$$

as the target y is the vector $f^* - f$ in our case. Consequently, for any neuron h_i, one has:

$$w_i^2 \leqslant \frac{1}{d_m^2} \mathcal{L}(f)\,. \tag{183}$$

As the norm of the neuron is $\|h_i\|^2 = w_i^2 \sum_j A_{ji}^2$, one still has to bound the activities $A_{ji} = \mathrm{ReLU}(a \cdot x_j - b_i)$. As a was chosen a unit direction, the values $a \cdot x_j$ span a domain smaller than the diameter of the dataset $\mathcal{D}$: $|a \cdot (x_j - x_{j'})| \leqslant \|x_j - x_{j'}\| \leqslant \mathrm{diam}(\mathcal{D})\ \forall j, j'$. Hence all values $\forall i, j, |A_{ij}| = |a \cdot x_i - b_j| = |a \cdot x_i - a \cdot x_j + d_m| < d_M := \mathrm{diam}(\mathcal{D}) + d_m$. Note that d_M depends only the dataset geometry, as for d_m.

We now have:

$$\|h_i\|^2 = w_i^2 \sum_j A_{ji}^2 \leqslant n \frac{d_M^2}{d_m^2} \mathcal{L}(f) \tag{184}$$

which ends the proof.

For higher output dimensions d', one vector w of output weights is estimated per dimension, independently, leading to the same bound for each dimension. The square norms of neurons are summed over all dimensions and thus multiplied by at most d'.

D.6 Reaching 0 training error in n neuron additions by overfitting each dataset sample in turn

If one allows updating already existing output weights at the same time as one adds new neurons, then it is possible to reach 0 training error in only n steps (where n is the size of the dataset) while decreasing the loss at each addition.

This scenario is closer to the one we consider with TINY, as we compute the optimal update of existing weights inside the layer, as a byproduct of new neuron estimation, and apply them.

However the existence proof here follows a very different way to create new neurons, taylored to obtain a constructive proof, and inspired by the previous section. See Appendix D.7 for another, more generic proof, applicable to a wide range of growth methods.

Here we consider the same approach as in Appendix D.5 above, but introducing neurons one by one instead of n neurons at once. After computing a and the biases b_j, thus forming the activity matrix A, we add only the last neuron h_n. The activity of this neuron is 0 for all input samples x_j except for the last one, for which it is $A_{nn} > 0$. Thus, the neuron h_n separates the sample x_n from the rest of the dataset, and it is easy to find w_n so that the loss gets to 0 on that training sample, without changing the outputs for other samples.

Similarly, one can then add neuron h_{n-1}, which is active only for samples x_{n-1} and x_n. However designing w_{n-1} so that the loss becomes 0 at point x_{n-1} disturbs the output for point x_n (and for that point only). Luckily if one allows to update w_n then there exists a (unique) solution (w_{n-1}, w_n) to achieve 0 loss at both points. This is done exactly as previously, by solving $y = Aw$, but considering only the 2 last lines and rows of A, leading to a smaller 2×2 system which is also lower-triangular with positive diagonal.

Proceeding iteratively this way adds neuron one by one in a way that sends each time one more sample to 0 loss. Thus adding n neurons is sufficient to achieve 0 loss on the full training set, and this in a way that each time decreases the loss.

Note that updating existing output weights w_i while adding a new neuron, to decrease optimally the loss, is actually what TINY does. However, the construction in this Appendix completely overfits each sample in turn, by design, without being to generalize to new test points. On the opposite, TINY exploits correlations over the whole dataset to extract the main tendencies.

D.7 TINY reaches 0 training error in n neuron additions

We will now show that the TINY approach, as well as any other suitable greedy growth method, implemented within the right optimization procedure, reaches 0 training error in at most n steps (where n is the size of the dataset), almost surely.

Before stating it formally, we need to introduce the optimization protocol, growth completion and a probability measure over activation functions.

Optimization protocol. For this we consider the following optimization protocol conditions, that has to be applied at least during the last, n-th addition step:

- a **full batch** approach,
- when adding new neurons, also compute and add the **optimal moves of already existing parameters** (i.e. of output weights w).

The first point is to ensure that all dataset samples will be taken into account in the loss during the n-th update. Otherwise, for instance if using minibatches instead, the optimization of output weights w will not be able to overfit the training loss.

The second point is to make sure that, after update, the output weights w will be optimal for the training loss. Note that in the mean square regression case, this is easy to do, as the loss is quadratic in w: the optimal move (leading to the global optimum f^*) can be obtained by line search over the natural gradient (which is obtained for free as a by-product of TINY's projection of $\boldsymbol{V}_{\text{goal}}$, and is proportional to $f^* - f$). This is precisely what we do in practice with TINY when training networks (except when comparing with other methods and using their own protocol).

Growth completion. For this proof to make sense, we will need the growth method to actually be able to perform n neuron additions, if it has not reached 0 training loss before. A counter-example would be a growth method that gets stuck at a place where the training loss is not 0 while being unable to propose new neuron to add. In the case of TINY, this can happen when no correlation between inputs x_i and desired output variations $f^*(x_i) - f(x_i)$ can be found anymore. To prevent this, one can choose any auxiliary method to add neurons in such cases, for instance random directions, solutions of higher-order expressivity bottleneck formulations using further developments of the activation function, or locally-optimal neurons found by gradient descent. Some auxiliary methods are guaranteed to further decrease the loss by a neuron addition (cf. Appendices D.2, D.3, D.4), while any other one will be guaranteed not to increase the loss if combined with a line search along that neuron direction.

We will name *completed-TINY* the completion of TINY by any such auxiliary method.

Activation function. For technical reasons, the result will stand *almost surely* only, depending on the invertibility of a certain matrix, namely, the activation matrix A, defined as $A_{ij} = \sigma(\boldsymbol{v}_j \cdot \boldsymbol{x}_i + b_j)$, indexed by samples i and neurons j.

Generally speaking, kernels induced by neurons $k_j : \boldsymbol{x} \mapsto \sigma(\boldsymbol{v}_j \cdot \boldsymbol{x} + b_j)$ form free families, in the sense that the are linearly independent (to the notable exception of the linear kernel). This linear independency means that a linear combination of kernels cannot be equal, as a function, to another kernel with different parameters. Equality is to be understood as *for all possible points* $\boldsymbol{x}$ *ever*. However here we will evaluate the functions only at a finite number n of points (the dataset samples), therefore linear independence will be considered between the rows of the activation matrix A. This notion of linear dependence is much weaker: kernels might form a free family as functions but be linearly dependent once restricted to the dataset samples, by mere chance. While this is not likely (over dataset samples), this is not impossible in general (though of measure 0), and it is difficult to express an explicit, simple condition on the activation function to be sure that the activation matrix A is *always* invertible (up to slight changes of parameters). Thus instead we will express results *almost surely* over activation functions and neuron parameters.

For most activation functions in the space of smooth functions, the activation matrix A will be invertible almost surely over all possible datasets. In the unlucky case where the matrix is not invertible, an infinitesimal move of the neurons' parameters will be sufficient to make it invertible. For some activation functions, however, such as linear or piecewise-linear ones (e.g., ReLU), the matrix might remain non-invertible over a wide range of parameter variations (unless further assumptions are made on the neurons added by the growth process). Yet, in such cases, slight perturbations of the activation function (i.e., choosing another, smooth, activation function, arbitrarily close to the original one) will yield invertibility.

To properly define "*almost surely*" regarding activation functions, let us restrict the activation function σ to belong to the space $\mathfrak{P}$ of polynomials of order at least n^2, that is:

$$\sigma(x) \;=\; \sum_{k=0}^{K} \gamma_k \, x^k \tag{185}$$

with $n^2 \leqslant K < +\infty$, and non-0 highest-order amplitude $\gamma_K \neq 0$. This set $\mathfrak{P}$ is dense in the set of all continuous functions over the set $\Omega = [-r_M, r_M]^d$ which is an hypercube of sufficient radius r_M to cover all samples from the given dataset. One can define probability distributions over $\mathfrak{P}$, for instance consider the density $p(\sigma) = \frac{\alpha}{K^2} \prod_{k=0}^{K} \frac{e^{-\gamma_k^2}}{\sqrt{2\pi}}$ with a factor $\alpha = \left(\frac{\pi^2}{6} - \sum_{k<n^2} \frac{1}{k^2}\right)^{-1}$ to normalize the distribution, and where K is the order of the polynomial and thus depends on σ. This density is continuous in the space of parameters γ_k (though not continuous in usual functional metric spaces). Note that the decomposition of any $\sigma \in \mathfrak{P}$ as a finite-order polynomial is unique, as monomials of different orders are linearly independent.

We can now state the following lemma (that we will prove later):

Lemma D.1 (Invertibility of the activation matrix)**.** *Let $\mathcal{D} = \{x_i,\, 1 \leqslant i \leqslant n\}$ be a dataset of n distinct points, and let $\sigma : \mathbb{R} \to \mathbb{R}$ be a function in $\mathfrak{P}$, that is, a polynomial of order at least n^2. Then with probability 1 over function and neuron parameters (γ_k), $(\boldsymbol{v}_j)$ and (b_j), the activity matrix A defined by $A_{ij} = \sigma(\boldsymbol{v}_j \cdot \boldsymbol{x}_i + b_j)$ is full rank.*

and the following proposition:

Proposition D.4 (Reaching 0 training error in at most n neuron additions)**.** *Under the assumptions above (polynomial activation function of order $\geqslant n^2$, full batch optimization and computation of the optimal moves of already existing parameters), completed-TINY reaches 0 training error in at most n neuron additions almost surely.*

Proof. If the growth method reaches 0 training error before n neuron additions, the proof is done. Otherwise, let us consider the n-th neuron addition. We will show in Lemma D.1 that the activity matrix A, defined by $A_{ij} = \sigma(\boldsymbol{v}_j \cdot \boldsymbol{x}_i + b_j)$, indexed by samples i and neurons j, is invertible. Then there exists a unique $\boldsymbol{w} \in \mathbb{R}^n$ such that $A\boldsymbol{w} = f^*$, i.e. $\sum_j w_j \sigma(\boldsymbol{v}_j \cdot \boldsymbol{x}_i + b_j) = f^*(\boldsymbol{x}_i)$ for each point $\boldsymbol{x}_i$ of the dataset. This vector of output parameters $\boldsymbol{w}$ realises the global minimum of the loss over already existing weights: $\inf_{\boldsymbol{w}} \mathcal{L}(f_{\boldsymbol{v},\boldsymbol{w}}) = \inf_{\boldsymbol{w}} \|A\boldsymbol{w} - f^*\|^2$. They are also the ones found by a natural gradient step over the loss (up to a factor 2, that can easily be found by line search as the loss is convex). Then after that update the training loss is exactly 0. □

Note: piecewise-linear activation functions such as ReLU are not covered by this proposition. However the result might still hold with further assumptions over the growth process. For instance with the method in Zhang et al. (2017), the ReLU neurons are chosen in such a way that the matrix A is full rank by construction.

Proof of Lemma D.1. Let us first show that if, unluckily, for a given activation function σ and given parameters $(\boldsymbol{v}_j, b_j)$, the matrix A is not full rank, then upon infinitesimal variation of the parameters, the matrix A becomes full rank.

Indeed, if all pre-activities $a_{i,j} := \boldsymbol{v}_j \cdot \boldsymbol{x}_i + b_j$ are not distinct for all i, j, then an infinitesimal variation of the vectors $\boldsymbol{v}_j$ can make them distinct. For this, one can see that the set of directions $\boldsymbol{v}_j$ on which any two dataset points $\boldsymbol{x}_i$ and $\boldsymbol{x}_{i'}$ have the same projection is finite (since it has to be the direction of $\boldsymbol{x}_i - \boldsymbol{x}_{i'}$, for a given pair of dataset samples (i, i')) and thus of measure 0. As a consequence with probability 1 over neuron parameters $\boldsymbol{v}_j$ and b_j, all pre-activities are distinct.

Now, if the matrix A is not invertible, as invertible matrices are dense in the space of matrices, one can easily find an infinitesimal change δA to apply to A to make it invertible. This corresponds to changing the activation function σ accordingly at each of the n^2 distinct pre-activity values. Since σ has more than n^2 parameters, this is doable. For instance one can select the n^2 first parameters and search for a suitable variation $\boldsymbol{g} := (\delta\gamma_k)_{0 \leqslant k < n^2}$ of them by solving the linear system $S\,\boldsymbol{g} = \delta A$ where the $n^2 \times n^2$ matrix S is

defined by $S_{ij,k} = a_{i,j}^k = (\boldsymbol{v}_j \cdot \boldsymbol{x}_i + b_j)^k$. This matrix S is invertible because any $\boldsymbol{g}$ such that $S\,\boldsymbol{g} = 0$ would induce:

$$\forall i, j, \quad \sum_{k=0}^{n^2-1} \delta\gamma_k \; a_{i,j}^k = 0 \tag{186}$$

and thus the polynomial $P(x) = \sum_{k=0}^{n^2-1} \delta\gamma_k \; x^k$ has at least n^2 roots while being of order at most $n^2 - 1$. Thus $S\,\boldsymbol{g} = 0 \quad \Longrightarrow \quad \boldsymbol{g} = 0$ and S is invertible. Note that as δA is infinitesimal, $\boldsymbol{g} = S^{-1}\,\delta A$ will be infinitesimal as well, and so is the change brought to the activation function σ.

Consequently we have that the set of activation functions σ and neuron parameters $(\boldsymbol{v}_j, b_j)$ for which the matrix A is full rank is dense in the set of polynomials $\mathfrak{P}$ of order at least n^2 and of neuron parameters $\mathfrak{N}$.

Now, the function $\det : \mathfrak{P} \times \mathfrak{N} \to \mathbb{R}$, $((\gamma_k)_k, (\boldsymbol{v}_j, b_j)_j) \mapsto \det A = \det\left(\sigma_\gamma(\boldsymbol{v}_j \cdot \boldsymbol{x}_i + b_j)\right)$ is smooth as a function of its input parameters (the determinant being a polynomial function of the matrix coefficients). As this continuous function is non-0 on a dense set of its inputs, the pre-image $\det^{-1}\{0\}$ is closed and contains no open subset. This is not yet sufficient to prove that this pre-image is of measure 0 (e.g., fat Cantor set).

For a fixed order K, one can see this function as a polynomial of its inputs γ_k and $\boldsymbol{v}_j, b_j$, and conclude[3] that the set of its roots is of measure 0. As a consequence, the probability, over coefficients γ_k or equivalently over polynomials σ of order K, that $\det A$ is non-0, is 1. As this stands for all K, we have that the probability that the matrix A is invertible is at least the mass of polynomials of all orders K, i.e. $\sum_{k \geqslant n^2} \frac{\alpha}{k^2} = 1$. Thus A is invertible with probability 1. □

E Technical details

E.1 Batch size to estimate the new neuron and the best update

In this section we study the variance of the matrices $\boldsymbol{M}^*$and $\boldsymbol{S}^{-1/2}\boldsymbol{N}$ computed using a minibatch of n samples, seeing the samples as random variables, and the matrices computed as estimators of the true matrices one would obtain by considering the full distribution of samples. Those two matrices are the solutions of the multiple linear regression problems defined in (39) and in (51), as we are trying to regress the desired update noted Y onto the span of the activities noted X. We suppose we have the following setting :

$$Y \sim \boldsymbol{A}X + \varepsilon, \;\; \varepsilon \sim \mathcal{N}(0, \sigma^2), \;\; \mathbb{E}[\varepsilon|X] = 0 \tag{187}$$

where the (X_i, Y_i) are *i.i.d.* and A is the oracle for $\boldsymbol{M}^*$ or matrix $\boldsymbol{S}^{-1/2}\boldsymbol{N}$. If Y is multidimensional, the the total variance of our estimator can be seen as the sum of the variances of the estimator on each dimension of Y.

We now suppose that $Y \in \mathbb{R}$ and note $\hat{A} := \boldsymbol{Y}\boldsymbol{X}^T(\mathbf{X}\mathbf{X}^T)^+$ the solution of 187. We first remark that $\hat{\boldsymbol{A}}\boldsymbol{X} = \boldsymbol{Y}\boldsymbol{P}$ with $\boldsymbol{P} = \boldsymbol{X}^T(\boldsymbol{X}\boldsymbol{X}^T)^+\boldsymbol{X} \in \mathbb{R}(n, n)$. It follows that when $n \leq p$, almost surely we have $rk(\boldsymbol{P}) = n$ and $\boldsymbol{Y}\boldsymbol{P} = \boldsymbol{Y}$, resulting in a zero expressivity bottleneck for that specific mini-batch, ie $\boldsymbol{Y} = \hat{\boldsymbol{A}}\boldsymbol{X}$. In practice, we do not consider $n < p$ as the solution ($\delta\boldsymbol{W}$ or $\boldsymbol{A}, \boldsymbol{\Omega}$) would overfit a specific mini-batch and would increase the expressivity bottleneck for the rest of the dataset.

We now suppose that $n > p$, we now have almost surely that $rk(\boldsymbol{P}) = p$ and $\boldsymbol{Y}\boldsymbol{P} \neq \boldsymbol{Y}$. We now study the variance of the estimator $\hat{\boldsymbol{A}}$. We have almost surely that $\boldsymbol{X}\boldsymbol{X}^T$ is invertible and note $(\boldsymbol{X}\boldsymbol{X})^{-1}$ its inverse. Taking the expectancy on the variable ε we have $\mathrm{var}(\hat{A}) = \sigma^2(\mathbf{X}\mathbf{X}^T)^{-1}$. If n is large, and if matrix $\frac{1}{n}\mathbf{X}\mathbf{X}^T \to \boldsymbol{Q}$, with $\boldsymbol{Q}$ non singular, then, asymptotically, we have $\hat{\boldsymbol{A}} \sim \mathcal{N}(\boldsymbol{A}, \sigma^2\frac{\boldsymbol{Q}^{-1}}{n})$, which is equivalent to $(\hat{\boldsymbol{A}} - \boldsymbol{A})\frac{\sqrt{n}}{\sigma}\boldsymbol{Q}^{1/2} \sim \mathcal{N}(0, I)$. Then $||(\hat{\boldsymbol{A}} - \boldsymbol{A})\frac{\sqrt{n}}{\sigma}\boldsymbol{Q}^{1/2}||^2 \sim \chi^2(k)$ where k is the dimension of X. It follows that $\mathbb{E}\left[||(\hat{\boldsymbol{A}} - \boldsymbol{A})\boldsymbol{Q}^{1/2}||^2\right] = \frac{k\sigma^2}{n}$ and as $\boldsymbol{Q}^{1/2}\boldsymbol{Q}^{1/2^T}$ is positive definite, we conclude that $\mathrm{var}(\hat{\boldsymbol{A}}) \leqslant \frac{k\sigma^2}{n\lambda_{\min}(\boldsymbol{Q})}$.

[3]See for instance a proof by recurrence that roots of a polynomial are always of measure 0: `https://math.stackexchange.com/questions/1920302/the-lebesgue-measure-of-zero-set-of-a-polynomial-function-is-zero` .

In practice and to keep the variance of our estimators stable during architecture growth, for the estimation of the best neuron to add we use batch size

$$n \propto \frac{(SW)^2}{P},$$

with the notations defined in Figure 12, since the matrices we estimate have side size SW and that each input sample contains P values, i.e. P quantities that each play the role of X here.

E.2 Batch size for learning

We adjust the batch size for gradient descent as follow : the batch size is set to $b_{t=0} = 32$ at the beginning of each experiment, and it is scheduled to increase as the square root of the complexity of the model (ie number of parameters). If at time t the network has complexity C_t parameters, then at time $t+1$ the training batch size is equal to $b_{t+1} = b_t \times \sqrt{\frac{C_{t+1}}{C_t}}$.

E.3 Normalization

E.3.1 Figures 5 and 16 : Usual normalization

For the GradMax method of figure 5 and 16, before adding the new neurons to the architecture, we normalize the out-going weight of the new neurons according to Evci et al. (2022), ie :

$$\alpha_k^* \leftarrow 0 \tag{188}$$

$$\text{for 5} \quad \omega_k^* \leftarrow \omega_k^* \times \frac{1e-3}{\sqrt{||(\omega_j^*)_{j=1}^{n_d}||_2^2/n_d}} \tag{189}$$

$$\text{for 16} \quad \omega_k^* \leftarrow \omega_k^* \times \sqrt{\frac{1e-3}{||(\omega_j^*)_{j=1}^{n_d}||_2^2/n_d}} \tag{190}$$

For TINY method of both figures, the previous normalization process is mimicked by normalizing the in and out going weights by theirs norms and multiplying them by $\sqrt{1e-3}$, ie :

$$\alpha_k \leftarrow \alpha_k^* \times \sqrt{\frac{1e-3}{||(\alpha_j^*)_{j=1}^{n_d}||_2^2/n_d}} \tag{191}$$

$$\omega_k \leftarrow \omega_k^* \times \sqrt{\frac{1e-3}{||(\omega_j^*)_{j=1}^{n_d}||_2^2/n_d}} \tag{192}$$

E.3.2 Figure 8 : Amplitude Factor

For the Random and the TINY methods of figure 8, we first normalize the parameters as :

For the new neurons

$$\alpha_k^* \leftarrow \alpha_k^* \times \frac{1}{\sqrt{||(\alpha_j^*)_{j=1}^{n_d}||_2^2/n_d}}$$

$$\omega_k^* \leftarrow \alpha_k^* \times \frac{1}{\sqrt{||(\omega_j^*)_{j=1}^{n_d}||_2^2/n_d}}$$

For the best update

$$\boldsymbol{W}^* \leftarrow \boldsymbol{W}^* \times \frac{1}{\sqrt{||\boldsymbol{W}^*||_2^2/n_d}}$$

Then, we multiply them by the amplitude factor γ^* :

For the new neurons :

$$\alpha_k^*,\ \ \omega_k^* \ \leftarrow \alpha_k^*\gamma^*, \omega_k^*\gamma^*$$
$$\gamma^* := \underset{\gamma\in[-L,L]}{\arg\min} \sum_i \mathcal{L}(f_{\theta\oplus\gamma\theta_{\leftrightarrow}^K}(\boldsymbol{x}_i), \boldsymbol{y}_i)$$

For the best update :

$$\boldsymbol{W}_l^* \ \leftarrow \gamma^*\boldsymbol{W}_l$$
$$\gamma^* := \underset{\gamma\in[-L,L]}{\arg\min} \sum_i \mathcal{L}(f_{\theta+\gamma\boldsymbol{W}^*}(\boldsymbol{x}_i), \boldsymbol{y}_i)$$

Where the operation $\gamma\theta_{\leftrightarrow}^{K^*} = (\gamma\alpha_k^*, \gamma\omega_k^*)_k^K$ is the concatenation of the neural network with the new neurons and $\theta+\gamma\boldsymbol{W}^*$ is the update of one layer with its best update. The batch on which γ^* is computed is different from the one used to estimate the new parameters and its size is fixed to 1000 for all experiments.

E.4 Full algorithm

In this section we describe in detail the pseudo code to plot 5 and 8. The function NewNeurons(l), in Algorithm 2, computes the new neurons defined at Proposition 3.2 for layer l sorted by decreasing eigenvalues. The function BestUpdate(l), in Algorithm 4 computes the best update at Proposition 3.1 for layer l.

Algorithm 1: Algorithm to plot Figure 5 and 8.

```
for each method [TINY, MethodToCompareWith] do
    Start from neural network N with initial structure s ∈ {1/4, 1/64};
    while N architecture doesn't match ResNet18 do
        for d in {depths to growth} do
            θ↔^K* = NewNeurons(d, method = method) ;
            Normalize θ↔^K* according to E.3;
            Add the neurons at d ;
            Train N for Δt epochs ;
            Save model N and its performance ;
        end
    end
end
```

Algorithm 2: NewNeurons

Data: l, method $= TINY$
Result: Best neurons at l

if $method = TINY$ **then**
  $\boldsymbol{M} = \text{BestUpdate}(l+1)$;
  $\boldsymbol{S}, \boldsymbol{N} = \text{MatrixSN}(l-1, l+1, \boldsymbol{M} = \boldsymbol{M})$;
  Compute the SVD of $\boldsymbol{S} := U \Sigma U^T$;
  Compute the SVD of $\boldsymbol{U}\sqrt{\Sigma}^{-1}\boldsymbol{U}\boldsymbol{N} := \boldsymbol{A}\Lambda\boldsymbol{\Omega}$;
  Use the columns of $\boldsymbol{A}$, the lines of $\boldsymbol{\Omega}$ and the diagonal of Λ to construct the new neurons of Prop. 3.2;
else if $method = GradMax$ **then**
  $\boldsymbol{M} = None$;
  $_, N = \text{MatrixSN}(l-1, l+1, \boldsymbol{M} = \boldsymbol{M})$;
  Compute the SVD of $N^T N$;
  Use the eigenvectors to define the new out-going weights ;
  Set the new in-going weight to 0;
else if $method = Random$ **then**
  $(\alpha_k, \omega_k)_{k=1}^{n_d} \sim \mathcal{N}(0, Id)$;
end

Algorithm 3: MatrixSN

Data: p_1, p_2 (layer indexes), $\boldsymbol{M} = \text{None}$
Result: Construct matrices $\boldsymbol{S}$ and $\boldsymbol{N}$

Take a minibatch $\mathbf{X}$ of size $\propto \frac{(SW)^2}{P}$;
Propagate and backpropagate $\mathbf{X}$;
Compute $\mathbf{V}_{goal}$ at p_2, ie $-\frac{\partial \mathcal{L}^{tot}}{\partial \mathbf{A_{p_2}}}$;
if $\boldsymbol{M} \neq None$ **then**
  $\mathbf{V}_{goal} -= \boldsymbol{M}\boldsymbol{B}_{p_1}$
end
$\mathbf{S}, \mathbf{N} = \boldsymbol{B}_{p_1}{\boldsymbol{B}_{p_1}}^T, \boldsymbol{B}_{p_1}{\boldsymbol{V}_{\text{goal}}}^T$;

Algorithm 4: BestUpdate

Data: l, index of a layer
Result: Best update at l

Take a minibatch $\mathbf{X}$ of size $\propto \frac{(SW)^2}{P}$;
Compute $(\mathbf{S}, \mathbf{N})$ with MatrixSN(l, l);
$\boldsymbol{M} = N^T S^{-1}$;

E.5 Computational complexity

We estimate here the computational complexity of the above algorithm for architecture growth.

Theoretical estimate. We use the following notations:

- number of layers: L
- layer width, or number of kernels if convolutions: W (assuming for simplicity that all layers have same width or kernels)
- number of pixels in the image: P ($P = 1$ for fully-connected)
- kernel filter size: S ($S = 1$ if fully-connected)
- minibatch size used for standard gradient descent: M
- minibatch size used for new neuron estimation: M'
- minibatch size used in the line-search to estimate amplitude factor: M''
- number of classical gradients steps performed between 2 addition tentatives: T

Complexity, estimated as the number of basic operations, cumulated over all calls of the functions:

- of the standard training part: $TMLW^2SP$
- of the computation of matrices of interest (function MatrixSN): $LM'(SW)^2P$
- of SVD computations (function NewNeurons): $L(SW)^3$
- of line-searches (function AmplitudeFactor): $L^2M''W^2SP$

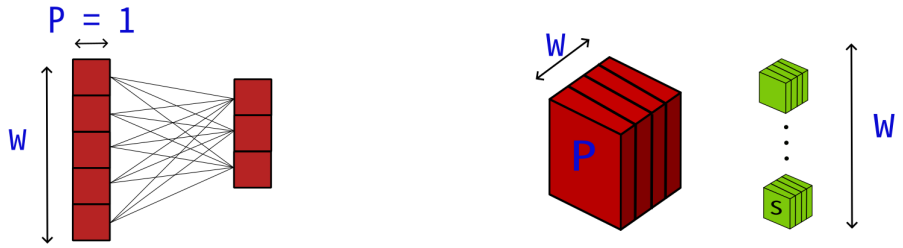

Figure 12: Notation and size for convolutional and linear layers

- of weight updates (function BestUpdate): LSW

The relative added complexity w.r.t. the standard training part is thus:

$$M'S\,/\,TM \;\;+\;\; S^2W\,/\,TMP \;\;+\;\; M''L\,/\,TM \;\;+\;\; 1\,/\,WTMP.$$

SVD cost is negligible. The relative cost of the SVD w.r.t. the standard training part is $S^2W\,/\,TMP$. In the fully-connected network case, $S = 1$, $P = 1$, and the relative cost of the SVD is then W/TM. It is then negligible, as layer width W is usually much smaller than TM, which is typically 10×100 for instance. In the convolutional case, $S = 9$ for 3×3 kernels, and $P \approx 1000$ for CIFAR, $P \approx 100000$ for ImageNet, so the SVD cost is negligible as long as layer width $W \ll 10000$ or 1 000 000 respectively. So one needs no worrying about SVD cost.

Likewise, the update of existing weights using the "optimal move" (already computed as a by-product) is computationally negligible, and the relative cost of the line searches is limited as long as the network is not extremely deep ($L < TM/M$").

On the opposite, the estimation of the matrices (to which SVD is applied) can be more ressource demanding. The factor $M'S/TM$ can be large if the minibatch size M' needs to be large for statistical significance reasons. One can show that an upper bound to the value required for M' to ensure estimator precision (see Appendix E.1) is $(SW)^2/P$. In that case, if $W > \sqrt{TMP/S^3}$, these matrix estimations will get costly. In the fully-connected network case, this means $W > \sqrt{TM} \approx 30$ for $T = 10$ and $M = 100$. In the convolutional case, this means $W > \sqrt{TMP/S^3} \approx 30$ for CIFAR and ≈ 300 for ImageNet. We are working on finer variance estimation and on other types of estimators to decrease M' and consequently this cost. Actually $(SW)^2/P$ is just an upper bound on the value required for M', which might be much lower, depending on the rank of computed matrices.

In practice. In practice the cost of a full training with our architecture growth approach is similar (sometimes a bit faster, sometimes a bit slower) than a standard gradient descent training using the final architecture from scratch. This is great as the right comparison should take into account the number of different architectures to try in the classical neural architecture search approach. Therefore we get layer width hyper-optimization for free.

F Additional experimental results and remarks

F.1 ResNet18 on CIFAR-100

Figures. In all plots the black line represents the average performance over two independent runs, and the colored regions indicate the confidence interval.

technical details of figure 5 and 8 The experiment were performed on 1 GPU. The optimizer is SGD($lr = 1e-2$) with the starting batch size 32 E.2. At each depth l we set the number n_l of neurons to be

added at this depth 2. These numbers do not depend on the starting architecture and have been chosen such that each depth will reach its final width with the same number of layer extensions. For the initial structure $s = 1/4$, resp. $1/64$, we set the number of layer extensions to 16, resp. 21, such that at depth 2 (named Conv2 in Table 3), $n_2 = (\text{Size}_2^{final} - \text{Size}_2^{start})/\text{nb of layer extensions} = (64-16)/16 = (64-1)/21 = 3$. The initial architecture is described in Table 3.

depth l	Conv2	Conv3	Conv5	Conv6	Conv8	Conv9	Conv11	Conv12
n_l	3	3	6	6	12	12	24	24

Table 2: Number of neurons to add per layer. The depth is identified by its name on tab 3

F.1.1 Performance for gradient-based methods

We present in table 5 two indicators of performance on classical visual datasets for four gradient-based methods: DART (Liu et al. (2019)), NORTH Maile et al. (2022), GradMax (Evci et al. (2022)), and TINY. We use the adjective *gradient-based* when the method uses the information from the propagation of the loss to search for an architecture. While GradMax, NORTH, and TINY increase the size of existing architectures (VGG and ResNet), DARTS uses cells to create the architecture, starting from a large graph of cells and removing what is considered unnecessary connections. We make the following remarks for the indicator $Time$, which is the time in GPU days to search for an architecture and train it, and the indicator *Acc.*, which is the accuracy on the test set:

- For NORTH, GradMax, and TINY, the accuracy on the test set is comparable, as all methods share a lot in terms of methodology, growth process, and neuron initialization (cf. B). Nonetheless, NORTH search time complexity is much larger as for each increase of architecture its strategy relies on trial-and-error methodology (going up to 1000 generations before actually increasing the architecture).
- The time search complexity of TINY will always be slightly higher than GradMax as it needs to project the desired update and compute the matrix $\boldsymbol{S}$(cf B).
- Although DARTS reaches good performances, its time search complexity is always greater than one GPU as it starts from a large architecture and decreases its size by removing connections between cells. For the ImageNet dataset, the indicator Time does not take into account the time spent searching for the basic cells, which are then used to create the graph.

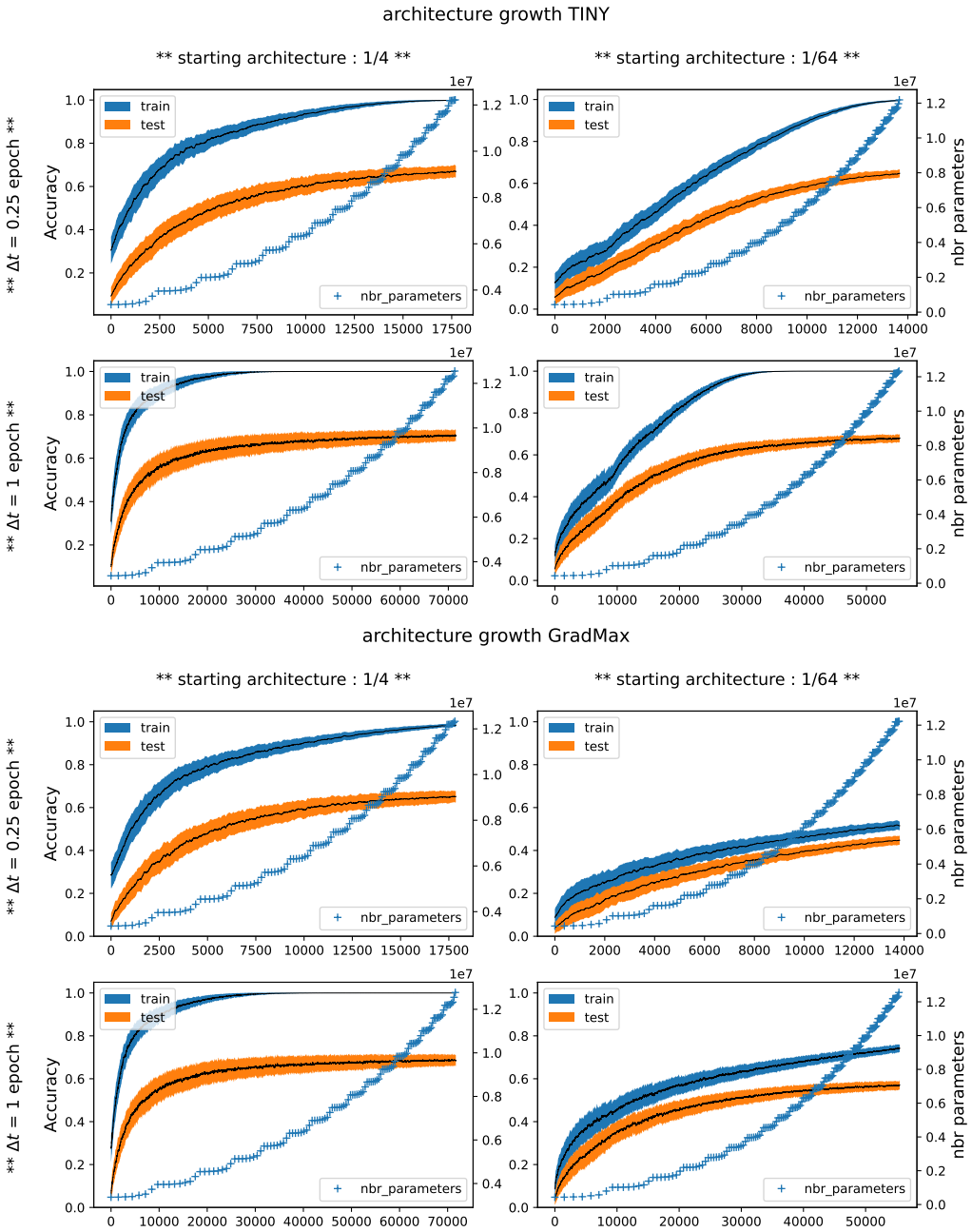

Figure 13: Accuracy and number of parameters during architecture growth for methods TINY and GradMax as a function of gradient step. Mean and standard deviation for four independent runs.

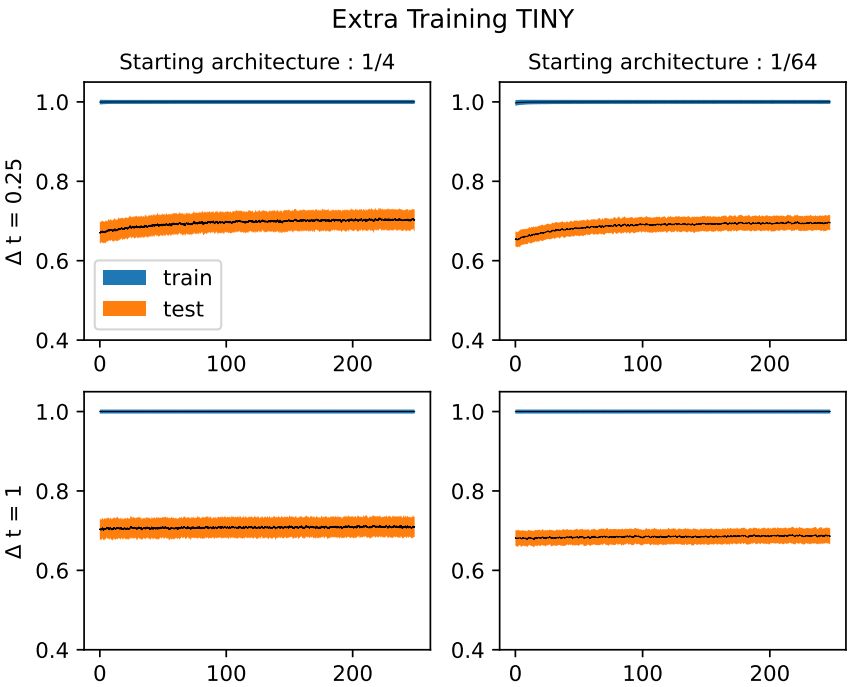

Figure 14: Accuracy as a function of the number of epochs during extra training for TINY on four independent runs.

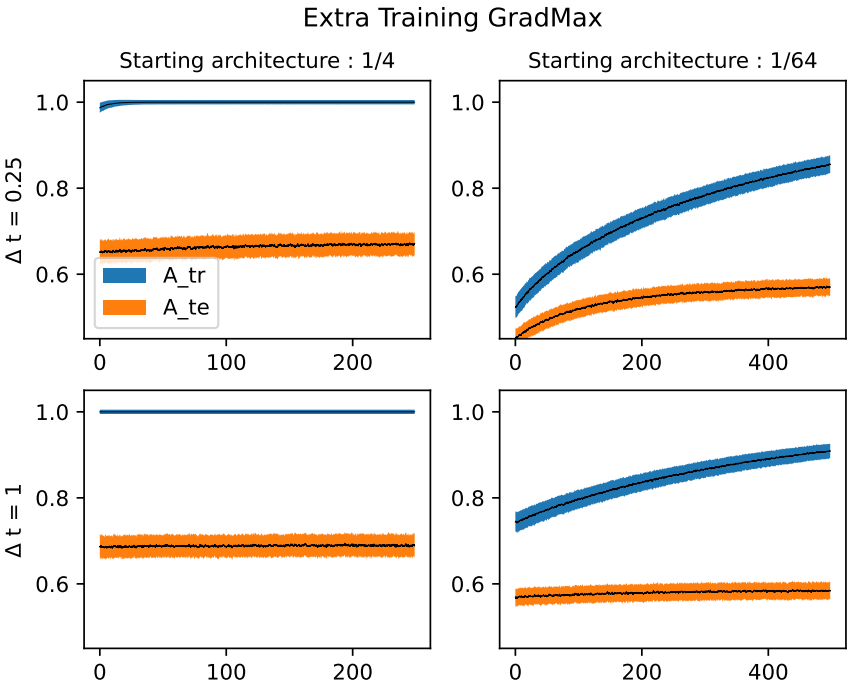

Figure 15: Accuracy curves as a function of the number of epochs during extra training for GradMax on four independent runs.

Table 3: Initial and final architecture for the models of Figure 5. Numbers in color indicate where the methods were allowed to add neurons (middle of ResNet blocks). In blue the initial structure for the model 1/64 and in green the initial structure for the model 1/4, ie 1/16 indicates that the model 1/64 started with 1 neuron at this layer while the model 1/4 started with 16 neurons at the same layer.

ResNet18			
Layer name	Output size	Initial layers (kernel=(3,3), padd.=1)	Final layers (end of Fig 5)
Conv 1	$32 \times 32 \times 64$	$3 \times 3,$	$3 \times 3, 64$
Conv 2	$32 \times 32 \times 64$	$3 \times 3, 64$ $3 \times 3, 1/16$ $3 \times 3, 1/16$ $3 \times 3, 64$	$3 \times 3, 64$ $3 \times 3, 64$ $3 \times 3, 64$ $3 \times 3, 64$
Conv 3	$32 \times 32 \times 64$	$3 \times 3, 64$ $3 \times 3, 1/16$ $3 \times 3, 1/16$ $3 \times 3, 64$	$3 \times 3, 64$ $3 \times 3, 64$ $3 \times 3, 64$ $3 \times 3, 64$
Conv 4	$16 \times 16 \times 64$	$3 \times 3, 128$	$3 \times 3, 128$
Conv 5	$16 \times 16 \times 128$	$3 \times 3, 128$ $3 \times 3, 2/32$ $3 \times 3, 2/32$ $3 \times 3, 128$	$3 \times 3, 128$ $3 \times 3, 128$ $3 \times 3, 128$ $3 \times 3, 128$
Conv 6	$16 \times 16 \times 128$	$3 \times 3, 128$ $3 \times 3, 2/32$ $3 \times 3, 2/32$ $3 \times 3, 128$	$3 \times 3, 128$ $3 \times 3, 128$ $3 \times 3, 128$ $3 \times 3, 128$
Conv 7	$8 \times 8 \times 256$	$3 \times 3, 256$	$3 \times 3, 256$
Conv 8	$8 \times 8 \times 256$	$3 \times 3, 256$ $3 \times 3, 4/64$ $3 \times 3, 4/64$ $3 \times 3, 256$	$3 \times 3, 256$ $3 \times 3, 256$ $3 \times 3, 256$ $3 \times 3, 256$
Conv 9	$8 \times 8 \times 256$	$3 \times 3, 256$ $3 \times 3, 4/64$ $3 \times 3, 4/64$ $3 \times 3, 256$	$3 \times 3, 256$ $3 \times 3, 256$ $3 \times 3, 256$ $3 \times 3, 256$
Conv 10	$4 \times 4 \times 512$	$3 \times 3, 512$	$3 \times 3, 512$
Conv 11	$4 \times 4 \times 512$	$3 \times 3, 512$ $3 \times 3, 8/128$ $3 \times 3, 8/128$ $3 \times 3, 512$	$3 \times 3, 512$ $3 \times 3, 512$ $3 \times 3, 512$ $3 \times 3, 512$
Conv 12	$4 \times 4 \times 512$	$3 \times 3, 512$ $3 \times 3, 8/128$ $3 \times 3, 8/128$ $3 \times 3, 512$	$3 \times 3, 512$ $3 \times 3, 512$ $3 \times 3, 512$ $3 \times 3, 512$
AvgPool2d	$1 \times 1 \times 512$		
FC 1	100	512×100	512×100
SoftMax	100		

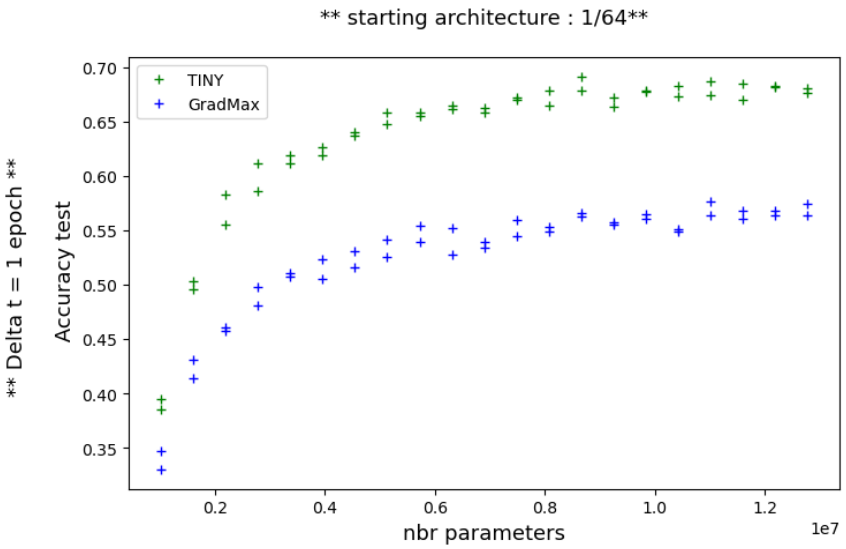

Figure 16: Accuracy on test of as a function of the number of parameters during architecture growth from $\text{ResNet}_{1/64}$ to ResNet18. **The normalization for GradMax is** $\sqrt{10^{-3}}$

	$\Delta t = 1$		Baseline
	TINY	GradMax	
s = 1/64	68.1 ± 0.5 68.7 ± 0.6 ^{5*}	57.2 ± 0.3 57.7 ± 0.3 ^{3*}	72.8 ± 0.3 ^{5*}

Table 4: Final accuracy on test of ResNet18 of 16 after the architecture growth (*grey*) and after convergence (*black*). The number of start indicated the multiple of 50 epochs needed to achieve convergence. With the starting architecture $\text{ResNet}_{1/64}$ and $\Delta t = 1$ the method TINY achieves 68.1 ± 0.5 on test after its growth and it reaches 68.7 ± 0.6 ^{5*}after $* := 5 \times 50$ epochs.

Method		Indicators			Dataset
		Arch.	Time	Acc.	
DARTS		$\times$	4	97.0 ± 0.1	CIFAR-10
		$\times$	6.5	97.8 ± 0.1	
		$\times$	4	73.3	ImageNet
NORTH		VGG-11	3.3	~ 76.4	CIFAR-10
		ResNet-28	0.4	~ 83.3	
		VGG-11	1.6	~ 55.7	CIFAR-100
GradMax†	$\Delta t = 0.25, s = 1/64$	ResNet-18	0.12	45.0 ± 0.4	CIFAR-100
	$\Delta t = 1, s = 1/64$	ResNet-18	0.26	56.8 ± 0.2	
TINY†	$\Delta t = 0.25, s = 1/64$	ResNet-18	0.13	65.8 ± 0.1	
	$\Delta t = 0.1, s = 1/64$	ResNet-18	0.28	68.1 ± 0.5	

Table 5: *Time* : GPU days for search the architecture and train it. *Acc.* : accuracy on test (%). $\dagger$: estimated with our implementation. For the NORTH method, we used the average performance by hand on their different initialization and strategies using their figure 4.