Spotting Expressivity Bottlenecks and Fixing Them Optimally

Manon Verbockhaven, Guillaume Charpiat
TAU team, LISN, INRIA Saclay / Université Paris-Saclay
prenom.nom@inria.fr

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 possible 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 the backpropagation. To do this, we propose a new mathematically well-grounded method to detect expressivity bottlenecks on the fly and solve them by adding suitable neurons when and where needed. Thus, while the standard approach requires large networks, in terms of number of neurons per layer, for expressivity and optimization reasons, we are able to start with very small neural networks and let them grow appropriately. As a proof of concept, we show results on the MNIST 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) 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 issues (Hanin & Rolnick, 2019b; Raghu et al., 2017), but then consumes extra computational power to train such big models, and often needs to reduce the model afterwards, possibly using probabilistic edges (Liu et al., 2019). Or one tries different architectures and keeps the most suitable one (in terms of performance-size compromise for instance), which multiplies the computational power by the number of trials. This latter approach relates to the Auto-DeepLearning field, 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; Miikkulainen et al., 2017), that 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 \textit{on the fly} during training, in a single run (no re-training), using first-order derivatives only, while avoiding neuron redundancy.

\textbf{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, Wu et al. (2019) and 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. Another example is GradMax (Evci et al., 2022), which seeks to minimize the loss as fast as possible and involves another quadratic problem. However, the neurons added by these approaches are possibly redundant with existing neurons, in particular 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. On the opposite we will explicitly take redundancy into account in our growing criterion.

\textbf{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 optimizers than small ones (Jacot et al., 2018). Typical, small networks exhibit issues with spurious local minima, while wide ones usually find good nearly-global minima. One of our goals is to train small networks without suffering from such optimization difficulties.

\textbf{Notions of expressivity.} Several concepts of expressivity or complexity exist in the Machine Learning literature, ranging from Vapnik-Chervonenkis dimension and Rademacher complexity 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); this quantity is difficult to estimate though. Also relevant and from Information Theory, the Minimum Description Length paradigm and Kolmogorov complexity enable to search for a compromise 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:

- take a functional analysis viewpoint over gradient descent on neural networks, suggesting to attempt to follow the functional gradient. We optimize not only the weights of the current architecture, but also the architecture itself on the fly, in order to progressively move towards more suitable parameterized functional spaces.
- properly define and quantify the notion of expressivity bottlenecks, globally at the neural network output as well as at each layer, and this in an easily computable way. This allows to localize the expressivity bottlenecks, by spotting layers with great lacks of expressivity.
- mathematically define the best possible neurons to add to a given layer to decrease lacks of expressivity as a quadratic problem; compute them and their associated expressivity gain.
- check that adding these best neurons is better indeed than adding random ones.
- are able to train a neural network without gradient descent (yet still relying on backpropagation) by just adding such best neurons, without parameter update.
- naturally obtain a series of compromises between performance and number of neurons, in a single run, thus removing the need for layer width hyper-optimization, and this in competitive computational complexity with respect to classically training a large model just once. One could define a target accuracy and stop adding neurons when it is reached.

\section{Notations and Definitions}

\subsection{Notations}

We consider a feedforward neural network with \( L \) hidden layers, \( f_\theta : \mathbb{R}^p \rightarrow \mathbb{R}^d \), where the parameters
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:

\[
\frac{df}{dt} = -\nabla_f \mathcal{L}(f) \quad \text{for some metric on the functional space } \mathcal{F} \quad (\text{typically, } L_2), \nabla_f \text{ denotes the functional gradient.}
\]

For a chosen \( \eta > 0 \), the descent direction \( \mathbf{v}_{\text{goal}} := -\eta \nabla_f \mathcal{L}(f) \) is a function of the same type as \( f \), indicating the best infinitesimal variation to add to \( f \) to decrease the loss.

In practice, to compute the gradient, a finite-dimensioned parametric space of functions is considered, a parametric search space \( \mathcal{A} \) by choosing a particular neural network architecture \( \theta \). The associated parametric search space \( \mathcal{F}_\mathcal{A} \) then consists of all possible functions \( f_\theta \) that can represented with such a network for any parameter value \( \theta \).

**Gradient descent reminder.** For the sake of simplicity, let us consider a loss of the form \( \mathcal{L}(f) = \mathbb{E}_{(x,y) \sim D} \left[ \mathcal{L}(f(x), y) \right] \). Under standard weak assumptions (A.1), and up to a multiplicative learning rate, the gradient descent is then of the form:

\[
\frac{\partial \theta}{\partial t} = -\nabla_\theta \mathcal{L}(f_\theta) = - \mathbb{E}_{(x,y) \sim D} \left[ \nabla_\theta \mathcal{L}(f_\theta(x), y) \right]
\]

Using the chain rule, this yields a function change:

\[
\mathbf{v}_{\text{GD}} := \eta \frac{\partial f_\theta}{\partial \theta} = \eta \frac{\partial f_\theta}{\partial \theta} \frac{\partial \theta}{\partial t} = \eta \frac{\partial f_\theta}{\partial \theta} \mathbb{E}_{(x,y) \sim D} \left[ \frac{\partial f_\theta}{\partial \theta}^T(x) \mathbf{v}_{\text{goal}}(x) \right]
\]

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

\[
\mathcal{T}_\mathcal{A}^{f_\theta} := \left\{ \frac{\partial f_\theta}{\partial \theta} \delta \theta \mid \text{s.t. } \delta \theta \in \Theta \right\}.
\]

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

\[
\mathbf{v}^* = \text{Proj}_{\mathcal{T}_\mathcal{A}}(\mathbf{v}_{\text{goal}})
\]

Figure 2: The expressivity bottleneck is measured as the difference between the optimal functional move \( \mathbf{v}^* \) given the architecture \( \mathcal{A} \) and the functional gradient \( \mathbf{v}_{\text{goal}} \). The former is the projection of the latter onto the tangent space \( \mathcal{T}_\mathcal{A} \).
Lack of expressivity. When \(-\eta \nabla f_\theta \mathcal{L}(f_\theta)\) does not belong to the reachable subspace \(T_A\), there is a lack of expressivity, that is, the parametric space \(A\) is not rich enough to follow the ideal functional gradient descent. This happens frequently with small neural networks.

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 \((x_0, x_1, x_2, x_3) = (0, 1.5, \pi, 4.5)\), 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 \(\Delta f_{\text{predict}}(x_i) := f_{\text{predict}}(x_i) - f_{\text{predict}}(x_i)\) goes into the same direction as minus the functional gradient \(v_{\text{goal}}(x_i) := -\eta \nabla f_{\text{predict}}(x_i) \mathcal{L}(f_{\text{predict}}(x_i), y_i) = -\eta 2(f_{\text{predict}}(x_i) - y_i)\) where \(y_i = f_{\text{true}}(x_i)\).

2.3 UPDATING LAYER ACTIVITIES

Ideal updates. The same reasoning can be applied to the pre-activations \(a_i\), seen as functions defined over the input space of the neural network. The optimal update of the weights of the different layers is then the projection of the desired update direction of pre-activation functions, i.e. \(\eta \nabla a_i \mathcal{L}(f_\theta)\), obtained by back-propagation, onto the linear subspace \(T_A^{a_i}\) of possible variations within the architecture, as we will detail now just like in the intuition part. Training a neural network is usually done by gradient descent, which consists in updating the weight matrices \(W_l\) recursively using back-propagation. More precisely, the gradient of the loss w.r.t. \(W_{i,j}^l\) is the product of the backpropagation from the loss \(\mathcal{L}\) till layer preactivations \(a_i(x)\) and of the layer-specific derivative from these preactivations till the weight in question: \(\frac{\partial a_i(x)}{\partial W_{i,j}^l} \nabla \mathcal{L}(\sigma_L(W_L(...(\sigma_i(u)))) | u = a_i(x)\). The opposite of the second term times \(\eta\) the positive real number, \(v_{\text{goal}}^i(x) := -\eta \nabla a_i \mathcal{L}(\sigma_L(W_L(...(\sigma_i(u)))) | u = a_i(x)\), indicates the desired update direction for \(u = a_i(x)\). Mathematically speaking, if at time \(t\) any activity update were possible at each layer \(l\), we would choose at \(t + 1\) the pre-activation function updates such that for all samples \(i = 1, ..., n:\)

\[
\Delta a_i(x_i) := a_i^{t+1}(x_i) - a_i^t(x_i) = -\eta \nabla a_i(x_i) \mathcal{L}(\sigma_L(W_L(...(a_i^t(x_i)))), y_i) | (x, y) = (x_i, y_i) \tag{1}
\]

Unfortunately, most of the time no parameter move \(\delta \theta\) is able to induce this progression for each \(x_i\) at the same time, because the \(\theta\)-parameterized family of functions \(a_i\) is not expressive enough. Intuitively the ideal update above does not result from a parameter update but from an update of the pre-activation functions in their functional spaces. Following this ideal update rather than the classical gradient descent would optimally decrease the loss, with an order-1 effect in \(\eta\) provided the gradient is non-0. Unlike this ideal update, the classical gradient descent decreases the global loss at first order in \(\eta\) but does not necessarily improve the prediction for all \(x_i\).

For the rest of the paper we will note \(v_{\text{goal}}^i(x_i) := -\eta \nabla a_i(x) \mathcal{L}(f_\theta(x), y)(|(x, y) = (x_i, y_i)\).

Activity update resulting from a parameter change. Given a subset of parameters \(\tilde{\theta}\), an incremental direction \(\delta \tilde{\theta}\) to update \(\tilde{\theta}\), and an amplitude \(\eta > 0\), the impact of the parameter update \(\delta \tilde{\theta}\) on the pre-activity \(a_i\) at layer \(l\) at order 1 in \(\delta \tilde{\theta}\) is \(v^i(x_i, \delta \tilde{\theta}) := \frac{\partial a_i(x)}{\partial \tilde{\theta}} \delta \tilde{\theta}\).

Remark 1. We could have choose to study the desired update for \(b_i\), our choice for \(a_i\) is explained in section [3,2].

3 EXPRESSIVITY BOTTLENECKS

We now define expressivity bottlenecks based on the activity updates, both actual \(v^i(\cdot)\) and desired \(v_{\text{goal}}^i(\cdot)\) ones (cf Figure 2):
We note this modification of architecture

\[ \theta \]

The added neurons could be chosen randomly, as in usual neural network initialisation, but this would

\[ \text{modified, while its gradient w.r.t.} \]

\[ \text{gradient only for the last part (projector).} \]

\[ \text{tor} \]

that is to say onto the span of all possible directions from post-activation, through the projec-

\[ \text{tion B} \]

\[ \text{where} \]

\[ ||| \]

\[ \text{Definition 3.1} \]

(Lack of expressivity) For a neural network \( f_0 \) and a minibatch \( \{(x_i, y_i)\}_{i=1}^n \), we define the lack of expressivity at layer \( l \) as how far the desired activity update \( V_{\text{goal}}^l \) is from the closest possible activity update realizable by a parameter change \( \delta \theta \):

\[
\min_{\delta \theta \in M(W_l, \text{shape})} \left\| V^l(X, \delta \theta) - V_{\text{goal}}^l(X) \right\|_{T \text{r}}^2
\]

(2)

where \( ||| \) stands for the \( L_2 \) norm and \( ||| \) \( T \text{r} \) for the Frobenius norm.

In the two following parts we fix a minibatch \( \{(x_i, y_i)\}_{i=1}^n \), i.e. a subset of the full dataset \( \mathcal{D} \). As \( X := [x_1, ..., x_1] \) is then fixed, we simplify the notation: \( A_l := A_l(X), \ B_l := B_l(X), \ V^l := V^l(X) \) ...

3.1 Best Move Without Modifying the Architecture of the Network

Let \( \delta W_l^* \) be the solution of (2) when the parameter variation \( \delta \theta \) is restricted to involve only layer \( l \) parameters, i.e. \( W_l \). 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:

\[
\delta W_l^* = \arg \min_{\delta W_l \in M(W_l, \text{shape})} \left\| V^l(\delta W_l) - V_{\text{goal}}^l \right\|_{T \text{r}}^2
\]

(3)

where \( M(k, l) \) is the set of matrices of size \( k \times l \), i.e. here of the size of \( W_l \). We denote by \( V_{0}^{l*} \) the associated activity variation:

\[
V_{0}^{l*} = \delta W_l^* B_{l-1} \quad v_{0}^{l*}(x_i) = \delta W_l^* b_{l-1}(x_i)
\]

Proposition 3.1. The solution of (3) and its associated activity variation are:

\[
\delta W_l^* = \frac{1}{n} V_{\text{goal}}^l B_{l-1}^T \left( \frac{1}{n} B_{l-1} B_{l-1}^T \right)^+ 
\]

where \( P^+ \) denotes the generalized inverse of matrix \( P \).

This update \( \delta W_l^* \) is not equivalent to the usual gradient descent update, whose form is \( \delta W_l^{\text{GD}} \propto V_{\text{goal}}^l B_{l-1}^T \). In fact \( V_{0}^{l*} \) is the projection of \( V_{\text{goal}}^l \) on the post-activation matrix of layer \( l-1 \), that is to say onto the span of all possible directions from post-activation, through the projector \( \frac{1}{n} B_{l-1} B_{l-1}^T \). 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 \( \delta W_l^* \) consists of a standard gradient \( (V_{\text{goal}}^l B_{l-1}^T) \) and of a (kind of) natural gradient only for the last part (projector).

3.2 Reducing Expressivity Bottleneck by Modifying the Architecture

To get as close as possible to \( V_{\text{goal}}^{l+1} \) and to increase the expressive power of the current neural network, we modify each layer of its structure. At layer \( l \), we add \( K \) neurons \( n_1, ..., n_K \) with input weights \( \alpha_1, ..., \alpha_K \) and output weights \( \omega_1, ..., \omega_K \) (cf Figure 4). We have the following change:

\[
W_l^T \leftarrow (W_l^T \alpha_1 \ ... \ \alpha_K) \quad \text{and} \quad W_{l+1} \leftarrow (W_{l+1} \omega_1 \ ... \ \omega_K).
\]

We note this modification of architecture \( \theta \leftarrow \theta \oplus \theta_{l+1}^K \) where \( \oplus \) is the concatenation sign and \( \theta_{l+1}^K := (\alpha_k, \omega_k)_{k=1}^K \) are the added neurons.

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

\[
\arg \min_{\theta_{l+1}^K, \delta W_l} \left\| V_{\text{goal}}^{l+1} - V^{l+1}(\delta W_l, \theta_{l+1}^K) \right\|_{T \text{r}}^2
\]
We define the matrices $V$ and note $\delta$ with $V$. Moreover for any number of neurons $K$, we have $\alpha_i \in \mathbb{R}^3$ et $\omega_i \in \mathbb{R}^3$ for $i = 1, 2$.

As the displacement $V^{l+1}$ at layer $l + 1$ is actually a sum of the moves induced by the neurons already present ($\delta W_l$) and by the added neurons ($\theta_{i+l+1}^K$), our problem rewrites as:

$$\arg \min_{\theta_{i+l+1}^K, \delta W_l} \left\| V_{\text{goal}}^{l+1} - V^{l+1}(\delta W_l) - V^{l+1}(\theta_{i+l+1}^K) \right\|_F^2$$ (4)

With $v^{l+1}(x, \theta_{i+l+1}^K) := \sum_{j=1}^{K} w_j (b_{l-1}(x)^T \alpha_k)$. Refering to the definition of $v^{l+1}(x)$ this choice have to be explained because the partial derivative with respect to $\alpha_k, \omega_k$ is actually 0 (See A.2). We solve this problem in two steps. Let us fix for the moment $\delta W_l \in \mathcal{M}(V_{\text{goal}}^{l+1}(x), |b_l(x)|)$, standing for an update of the matrix $W_{l+1}$, and search for the best new parameters $\theta_{i+l+1}^K$.

We note $V_{\text{goal proj}}^{l+1} = V_{\text{goal proj}}^{l+1}(\delta W_l) := V_{\text{goal}}^{l+1} - V^{l+1}(\delta W_l)$. We are looking for the following quantity:

$$\left( (\hat{\alpha}_k^*, \hat{\omega}_k^*)_{K^*}^{K^*}, K^* \right) := \arg \min_{(\alpha_k, \omega_k)_{K^*}^{K^*}} \left\{ \left\| V_{\text{goal proj}}^{l+1} - V^{l+1}(\theta_{i+l+1}^K) \right\|_F^2 \right\}$$ (5)

We define the matrices $N := 1/n B_{l-1}(V_{\text{goal proj}}^{l+1})^T$ and $S := 1/n B_{l-1}B_{l-1}^T$. Note that $N$ depends on $\delta W_l$. Using the low-rank matrix approximation theorem (Eckart & Young [1936]), we can solve this quadratic optimization problem as follows.

As $S$ is semi-positive definite, let us denote its Cholesky decomposition by $S = S^{1/2}S^{1/2}^T$, and consider the SVD of the matrix $S^{1/2}^{-1} N = \sum_{k=1}^{R} \lambda_k u_k v_k^T$ with $\lambda_1 \geq \ldots \geq \lambda_R \geq 0$, where $R$ is the rank of the matrix $N$. Then:

**Proposition 3.2.** The solution of (5) can be written as:

- optimal number of neurons: $K^* = R$
- their optimal weights: $(\hat{\alpha}_k^*, \hat{\omega}_k^*) = (S^{1/2}^T u_k, v_k)$ for $k = 1, \ldots, R$.

Moreover for any number of neurons $K \leq R$, and associated scaled weights $\hat{\theta}_{i+l+1}^{K^*}$, the expressivity gain and the first order in $\eta$ 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 $\lambda_k$:

$$\frac{1}{n} \left\| V_{\text{goal proj}}^{l+1} - V^{l+1}(\hat{\theta}_{i+l+1}^{K^*}) \right\|_F^2 = \frac{1}{n} \left\| V_{\text{goal proj}}^{l+1} \right\|_F^2 - \sum_{k=1}^{K} \lambda_k^2$$

$$\frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(f_{\theta_{i+l+1}^{K^*}}(x_i), y_i) = \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(f_{\theta_l}(x_i), y_i) + \frac{\sigma^2_l(0)}{\eta} \sum_{k=1}^{K} \lambda_k^2 + o(||\hat{\theta}_{i+l+1}^{K^*}||_2^2)$$

Figure 4: Adding two neurons at layer $l$ in purple ($K = 2$), with connections in purple. We have $\alpha_i \in \mathbb{R}^3$ et $\omega_i \in \mathbb{R}^3$ for $i = 1, 2$. 
Proposition 3.3. If $S$ is positive definite, then solving (5) is equivalent to taking $\omega_k = N \alpha_k$ and finding the $K$ first eigenvectors $\alpha_k$ associated to the $K$ largest eigenvalues $\lambda$ of the generalized eigenvalue problem:

$$NN^T \alpha_k = \lambda S \alpha_k$$

This formulation is useful when dimensions of $N$ and $S$ are large. Considering LOBPCG method (Peter Benner [29 Apr 2020]) allows not to invert the matrix and to compute the Cholesky factorization in proposition 3.2. In practice the matrix $S$ is positive definite except for $l - 1 = 0$, and even in this last case it is possible to define the Cholesky decomposition of $S$ (cf Appendix).

Corollary. The matrix $\delta W_l$ which minimizes (6) (through its impact on $V_{goal \leftrightarrow \text{proj}}^{l+1}$) or equivalently minimizes the sum of orders zero and one in $\eta$ of (7) is given by $\delta W_l^{l+1}$ in Proposition 3.1.

Corollary. For all integers $m, m'$ such that $m + m' \leq R$, at order one in $\eta$ it is equivalent to add $m + m'$ neurons simultaneously according to the previous method or to add $m'$ neurons then $m'$ neurons by applying successively the previous method twice.

Minimizing the distance (6), ie the distance between $V_{goal \leftrightarrow \text{proj}}^{l+1}(\delta W_l)$ and $V^{l+1}(\theta_{l+1})$, is equivalent to minimize the loss $\mathcal{L}$ at order one in $\gamma$, and it is directly due to the following development:

$$\mathcal{L}(f_{\theta \oplus \hat{\delta}^{K,*}}_{l+1}) = \mathcal{L}(f_{\theta}) - \sigma^2(0) \frac{1}{\eta} \sigma^2(0) \langle V_{goal \leftrightarrow \text{proj}}^{l+1}(\delta W_l), V^{l+1}(\theta_{l+1}) \rangle_{\text{Tr}} + o(||V^{l+1}(\theta_{l+1})||)$$

When solving (6), we notice that the family $\{V^{l+1}((\alpha_k, \omega_k))\}_{k=1}^K$ of pre-activity variations induced by adding the neurons $\hat{\delta}^{K,*}_{l+1}$ is orthogonal for the trace scalar product. We could say that the added neurons are orthogonal to each other in that sense. The addition of each neuron $k$ has an impact on the order of $\lambda_k$, which can be used to define a criterion to decide whether the neuron $k$ should be added to the layer or not, i.e., only if $\lambda_k^2 \mathcal{L}(f_{\theta}) > \tau$. We name the operation $\theta \leftarrow \theta \oplus \hat{\delta}^{K,*}_{l+1}$ the $K$-update of the network at layer $l$.

3.3 Choosing the amplitude factor $\gamma$ when adding neurons

Let us consider the $l+1$-th layer of the network. Having the best update for linear layer $l+1$, $\delta W_{l+1}$, and the best neurons to add accordingly to layer $l$, $(\hat{\alpha}_k, \hat{\omega}_k)_{k=1}^K$, we estimate the best factors $\gamma_0$ and $\gamma$ to multiply the new neurons with, in order to speed up the learning. Defining the updates $\theta^{(l+1)}(\gamma_0) = (W_1, ..., W_{l+1} + \gamma_0 \delta W_{l+1}, ..., W_L)$ and $\theta^{(l+1)}(\gamma) = (\gamma \hat{\alpha}_k, \gamma \hat{\omega}_k)_{k=1}^K$, we apply a line search algorithm to find a local minimum of the loss function:

$$\gamma_0^* := \arg \min_{\gamma \in \mathbb{V}(0)} - \frac{1}{n} \sum_{i=1}^n \mathcal{L}(f_{\theta^{(l+1)}(\gamma_0)}(X_i), Y_i)$$

$$\gamma^* := \arg \min_{\gamma \in \mathbb{V}(0)} - \frac{1}{n} \sum_{i=1}^n \mathcal{L}(f_{\theta^{(l+1)}(\gamma^*)}(X_i), Y_i)$$

where $\mathbb{V}(0)$ is a positive neighbourhood of zero.

Remark : the amplitude factor can also be defined differently, for example by choosing a different amplitude factor for the input and output parameters, i.e $(\sqrt{\eta} \hat{\alpha}_k, \sqrt{\eta} \hat{\omega}_k)_{k=1}^K$.

3.4 Variance of the estimator

Noting $n_l$ the number of neurons already in layer $l$ before the $K$-update, we discuss the variance of our estimate $\hat{\theta}^{K,*}_{l+1}$ for $l = 1, ..., L - 1$, as a function of the minibatch. At layer $l + 1$, the solution of (5) is an estimator of $\theta^{K,*}_{l+1} := (\hat{\alpha}_k, \hat{\omega}_k)_{k=1}^K$, which is the minimizer of:

$$\hat{\theta}^{K,*}_{l+1} := \arg \min_{\alpha, \omega} \mathbb{E} \left[ \left\| \hat{v}_{goal \leftrightarrow \text{proj}}^{l+1}(X_i) - \sum_{k=1}^K \hat{v}_k^{l+1}(X_i) \right\|^2 \right]$$

where the expectation is taken over all possible random samples $X_i \sim \mathcal{D}$. The variance of the directions $(\hat{\alpha}_k, \hat{\omega}_k)_{k=1}^K$ grows in $\sqrt{p}$ where $p := \max(n_{l-1}, n_{l+1})$ and decreases in $\sqrt{N}$ where $N$ is the minibatch size used for the estimation of $\hat{\theta}^{K,*}$. In practice we will start with $N = 100$ and increase $N$ with the architecture growth.
4 EXPERIMENTAL RESULTS

In Figures 5 and 6, we run an experiment on the MNIST dataset ([LeCun et al., 1998]), with 7 CPUs, repeated 20 times. All models are trained for 50 seconds with Adam(lr = 0.0001, µ = 0) with constant mini-batch of size 100. The activation functions are $\sigma_1 = \text{selu}$ if $l = 1, 2$ and $\sigma_3 = \text{Softmax}$. We plot accuracy mean and standard deviation on test set for our approach with architecture growth and for standard training with a fixed architecture. For our approach, we start with a feedforward model with 2 hidden layers of widths $[1, 1]$, initialized with Kaiming normal. Every 0.05 seconds (with Adam(lr = 0.0001)), we extend the two hidden layers according to our method. We compare the performance of our method with classical training of large models (Fig. 5), or of models with the same final architecture as ours (Fig. 6), to check performance when one already knows the correct architecture. Classic models are initialized with Kaiming normal. More graphics can be found in Appendix C. We note that large models take more computational time to train, and that architecture growth yields better or similar performance while avoiding layer width tuning.

Figure 5: All graphics represent the same experiment but from a different perspective. **Left**: after partitioning computational time on intervals of size 0.1 seconds, we compute a linear interpolation for the accuracy value. **Middle**: the accuracy value against number of epochs, where the time needed to compute the optimal neurons is not noticeable. **Right**: accuracy against computational time, where durations due to Cholesky decompositions and their happening instants are averaged over experiments, for better visualisation purposes.

Figure 6: **Left**: we plot the interpolation of accuracy on intervals of size 0.1 second. **Middle**: zoom of the top left plot. **Right**: the accuracy value against number of epochs, unlike middle plot in graphic 5, our method is trained on fewer epochs compared to classic model, indeed with equal architecture our method spends time computing the best neurons while the classic method continues its training.

5 ABOUT GREEDY GROWTH SUFFICIENCY

One might wonder whether a greedy approach on layer growth might get stuck in a non-optimal state. 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:

$$\inf_{f} \sum_{x \in D} ||f(x) - f^*(x)||^2$$
where \( f(x) \) is the output of the neural network and \( D \) is the training set.

**Proposition 5.1** (Greedy completion of an existing network). If \( f \) is not \( f^* \) yet, then 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. Furthermore:

**Proposition 5.2** (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 \( f \).

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. One can express a lower bound on how much the loss has improved (for the best such neuron), but it is not a very good bound without further assumptions on \( f \).

**Proposition 5.3** (Greedy completion by one infinitesimal neuron). The neuron in the previous proposition can be chosen to have arbitrarily small input weights.

This detail is important in that our approach is based on the tangent space of the function \( f \) and consequently manipulates infinitesimal quantities. Though we perform line search in a second step and consequently add non-infinitesimal neurons, our first optimization problem 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 \( \ref{eq:problem52} \) will not find neurons to add. Yet following Prop. \( \ref{prop:51} \) 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 \( \varepsilon \) on all of its input weights), for instance \( \sigma(\varepsilon \alpha \cdot x) \simeq \sigma(0) + \sigma'(0)\varepsilon \alpha \cdot x \pm \frac{1}{2} \sigma''(0)\varepsilon^2 (\alpha \cdot x)^2 + O(\varepsilon^3) \).

Though the linear term \( \alpha \cdot x \) might be uncorrelated over the dataset with desired output variation, i.e. \( \mathbb{E}_{x \sim D} \alpha \cdot x = 0 \), the quadratic term \( (\alpha \cdot x)^2 \), or higher-order ones otherwise, might be correlated. Finding neurons with such higher-order correlations can be done by increasing accordingly the power of \( (\alpha \cdot x) \) in the optimization problem \( \ref{eq:problem51} \). Note that one could consider other function bases that the polynomials from Taylor expansion. 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. The loss can then be improved, all the more with a line search to optimize the neuron amplitude.

However, such random neurons also contribute to other directions in the functional space than the desired one. This hinders the loss improvement expectable from them, as the line search will need to find a compromise with the loss changes brought by these other directions. This is confirmed experimentally in Appendix \( \ref{sec:app} \). To alleviate this, in the spirit of common neural network training practice, one could consider brute force combinatorics by adding many random neurons and hoping that one will be close enough to the desired direction. The difference with standard training is that we would perform such computationally-costly searches only when and where relevant, exploiting all simple information (linear correlations in each layer) first.

### 6 Conclusion

We have properly defined lacks of expressivity, and their minimization has allowed us to optimize the architecture on the fly, to better follow the functional gradient, enabling architecture growth. Apart from straightforward extension to other types of layers (such as convolutions) and to the addition of new layers, future work will pay attention to overfit possibilities (which we have not we have not observed so far, thanks to the optimally small number of parameters) and to neuron addition strategies.
REFERENCES


Xin Liang Peter Benner. Convergence analysis of extended lobpcg for computing extreme eigenvalues. 29 Apr 2020.


A Assumptions

A.1 Measure theory statement

Let $X$ be an open subset of $\mathbb{R}$, and $\Omega$ 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 $\omega$ for each $x \in X$.
- For almost all $\omega \in \Omega$, the partial derivative $f_x$ or $f$ accordind to $x$ exists for all $x \in X$.
- There is an integrable function $\theta : \Omega \to \mathbb{R}$ such that $|f_x(x, \omega)| \leq \theta(\omega)$ for all $x \in X$ and almost every $\omega \in \Omega$.

Then, for all $x \in X$,

$$\frac{d}{dx} \int_{\Omega} f(x, \omega) \, d\omega = \int_{\Omega} f_x(x, \omega) \, d\omega$$

See proof and details [mea]

A.2 Remarks

When increasing the size of layer $l$ with $b_{k+l+1}^l := (\alpha_k, \omega_k)^{K}_{k=1}$ starting with $(\alpha_k, \omega_k)^K_{k=1} = 0$, the outcome for $v^{l+1}(x, \theta^K_{l+l+1})$ is 0 because the gradient with respect to $(\alpha_k, \omega_k)^K_{k=1}$ is 0 in $(\alpha_k, \omega_k)^K_{k=1}$ is 0. In mathematical terms :

$$v^{l+1}(x, \theta^K_{l+l+1}) := \frac{\partial a_{l+1}(x)}{\partial (\alpha_k)^K_{k=1} |(\alpha_k)^K_{k=1} = 0} \theta^K_{l+l+1} = 0 \quad (9)$$

This is equivalent to say that for each family $(\omega_k)^{K}_{k=1}$, the tangent space in $a_{l+1}$ restricted to move in the family $(\alpha_k)^{K}_{k=1}$, ie $T_{A^{l+1}} := \{ \frac{\partial a_{l+1}}{\partial (\alpha_k)^K_{k=1} |(\alpha_k)^K_{k=1} = 0} |(\alpha_k)^K_{k=1} \in \{ R^{b_{l-1}(x)} \}^K \}$ varies with the family $(\omega_k)^{K}_{k=1}$, ie $T^{\omega_{l+1}} := T^{\omega_{l+1}}(\omega_k)^{K}_{k=1}$. Optimizing over the $\omega_k$ is equivalent to search for the better tangent space while optimizing on the $\alpha_k$ is equivalent to find the better projection on the tangent space defined by the $\omega_k$.

Note that making the derivative according to the $\alpha_k$ ease the problem by removing the non-linearity in $\sigma_i$. When reversing the roles of the $\alpha_k$ and of the $\omega_k$, ie fixing the $\alpha_k$ and compute $v^{l+1}(x, (\omega_k)^K_{k=1})$, it makes the problem harder to solve because the non linearity in $\sigma_l$ remains in the optimisation problem.

Taking an other point of view, you can consider the second order of $a_{l+1}(x)$ in $(\alpha_k, \omega_k)^K_{k=1}$ in 0 to recover the same expression. Indeed taking the Taylor expansion in $(\alpha_k, \omega_k)^K_{k=1}$

$$a_{l+1}(x) = a_{l+1}(x)|_{(\alpha_k, \omega_k)^K_{k=1}=0} + \sum_{k=1}^{K} \omega_k b_{l-1}(x)^T \alpha_k + o(||(\alpha_k)^K_{k=1}|| + ||(\omega_k)^K_{k=1}||)^2)$$

[TODO] A faire
Proof

Using the definition of the generalized inverse of \( M \)
Consider the function of the matrix \( N \)
If \( \lambda \) is positive definite, consider the Cholesky decomposition \( \hat{S} \), then the solution of (4) is written as \( R \).

\[
\delta W_t^* = \frac{1}{n} V_{\text{goal}}^T B_{t-1}^T \left( \frac{1}{n} B_{t-1} B_{t-1}^T \right)^+ \text{ and } V_0^* = \frac{1}{n} V_{\text{goal}}^T B_{t-1}^T \left( \frac{1}{n} B_{t-1} B_{t-1}^T \right)^+ B_{t-1}
\]

**Proof**
Consider the function \( g(\delta W_t) = ||V_{\text{goal}}^l - \delta W_t B_{t-1}||_F^2 \), then
\[
g(\delta W_t + H) = ||V_{\text{goal}}^l - \delta W_t B_{t-1} - H B_{t-1}||_F^2
\]
By identification \( \nabla_{\delta W_t} g(\delta W_t) = 2 (V_{\text{goal}}^l - \delta W_t B_{t-1}) B_{t-1}^T \)

\[
\nabla_{\delta W_t} g(\delta W_t) = 0 \implies V_{\text{goal}}^l B_{t-1}^T = \delta W_t B_{t-1} B_{t-1}^T
\]
Using the definition of the generalized inverse of \( M^+ \):

\[
\delta W_t^* = \frac{1}{n} V_{\text{goal}}^l B_{t-1}^T \left( \frac{1}{n} B_{t-1} B_{t-1}^T \right)^+
\]

B.2 Proposition 3.2

If \( S \) is positive definite, consider the Cholesky decomposition \( S = S^{1/2} S^{1/2}^T \), note \( R \) the rank of the matrix \( N \) and the SVD of the matrix \( S^{1/2}^{-1} N = \sum_{k=1}^{R} \lambda_k u_k v_k^T \) with \( 0 \leq \lambda_1 \leq \ldots \leq \lambda_R \) then the solution of (4) is written as \( R^* = R \) and \( (\tilde{\alpha}_k^*, \tilde{\omega}_k^*) = (S^{1/2}^{-1} u_k, v_k) \) for \( k = 1, \ldots, R_{t+1} \). Moreover for all \( K \leq R \) and \( \tilde{\beta}^{K,*}_{t+1} := (\tilde{\alpha}_k^*, \tilde{\omega}_k^*)_{k=1}^{K} \) and for \( \gamma \) positive, \( \tilde{\beta}^{K,*}_{t+1}(\gamma) :=

\((\gamma \alpha_k^*, \omega_k^*)\) for each \(k = 1, \ldots, K\), we have that:

\[
\frac{1}{n} \| V^{l+1}_{\text{goal proj}} - V^{l+1}\left(\hat{\theta}_{l+1}^K\right) \|^2_{\text{Tr}} = \frac{1}{n} \| V^{l+1}_{\text{goal proj}} \|^2_{\text{Tr}} - \sum_{k=1}^{K} \lambda_k^2
\]

(10)

\[
\frac{1}{n} \sum_{i=1}^{n} L(f_{\theta \oplus \hat{\theta}_{l+1}^K}(\gamma)(x_i), y_i) = \frac{1}{n} \sum_{i=1}^{n} L(f_{\theta}(x_i), y_i) - \sigma_l(0) \sum_{k=1}^{K} \lambda_k^2 + o(\gamma)
\]

(11)

**Proof**

\[
\arg \min_{\hat{\theta}_{l+1}^K} \left\{ \frac{1}{n} \| V^{l+1}_{\text{goal proj}} - V^{l+1}\left(\hat{\theta}_{l+1}^K\right) \|^2_{\text{Tr}} \right\} = \arg \min_{\hat{\theta}_{l+1}^K} \left\{ -\frac{2}{n} \langle V^{l+1}_{\text{goal proj}}, V^{l+1}\left(\hat{\theta}_{l+1}^K\right) \rangle_{\text{Tr}} + \frac{1}{n} \| V^{l+1}\left(\hat{\theta}_{l+1}^K\right) \|^2_{\text{Tr}} \right\}
\]

\[
= \arg \min_{\hat{\theta}_{l+1}^K} \frac{1}{n} g(\hat{\theta}_{l+1}^K)
\]

We note

\[
\frac{1}{n} g(\hat{\theta}_{l+1}^K) = -\frac{2}{n} \sum_{i} \sum_{k} v^{l+1}_{\text{goal proj}}(x_i)^T \alpha_k^T b_{l-1}(x_i) \omega_k
\]

\[
+ \frac{1}{n} \sum_{k,j} \sum_{i} \alpha_k^T b_{l-1}(x_i) \omega_k^T \omega_j \left( \alpha_j^T b_{l-1}(x_i) \right)
\]

\[
= -\frac{2}{n} \sum_{k} \alpha_k^T \left( \frac{1}{n} \sum_{i} b_{l-1}(x_i) v^{l+1}_{\text{goal proj}}(x_i)^T \right) \omega_k
\]

\[
+ \frac{1}{n} \sum_{k,j} \omega_k^T \omega_j \alpha_k^T \left( \frac{1}{n} \sum_{i} b_{l-1}(x_i) b_{l-1}(x_i)^T \right) \alpha_j
\]

\[
= -2 \sum_{k} \alpha_k^T N \omega_k + \sum_{k,j} \omega_k^T \omega_j \alpha_k^T S \alpha_j
\]

with \(N = \frac{1}{n} \sum_{i} b_{l-1}(x_i) v^{l+1}_{\text{goal proj}}(x_i)^T = \frac{1}{n} b_{l-1}(V^{l+1}_{\text{goal proj}})^T\),

\(S = \frac{1}{n} \sum_{i} b_{l-1}(x_i) b_{l-1}(x_i)^T = \frac{1}{n} b_{l-1} b_{l-1}^T\)
Suppose $S$ is semi definite positive and note $S = S^{1/2}S^{1/2T}$, $\gamma_k = S^{1/2T}\alpha_k$ and $S^{1/2-1}N = \sum_{r=1}^{R} \lambda_r v_r e_r^T$ the SVD of matrix $S^{1/2-1}N$.

\[-\sum_{k=1}^{K} \alpha_k^T N\omega_k = - \sum_{k}^{K} \gamma_k^T S^{1/2-1} N\omega_k\]

\[= - \sum_{k}^{K} \sum_{r=1}^{R} \gamma_k^T \lambda_r v_r e_r^T \omega_k\]

\[= - \text{Tr} \left( \sum_{k}^{K} \sum_{r=1}^{R} \lambda_r \left( \gamma_k^T v_r e_r^T \right) \omega_k \right)\]

\[= - \text{Tr} \left( \sum_{k}^{K} \sum_{r=1}^{R} \lambda_r \omega_k \gamma_k^T v_r e_r^T \right)\]

\[= - \left( \sum_{k}^{K} \gamma_k^T \omega_k, \sum_{r=1}^{R} \lambda_r v_r e_r^T \right)_{\text{Tr}} \text{ with } \langle A, B \rangle_{\text{Tr}} = \text{Trace}(A^T B)\]

\[\sum_{k,j}^{K} \omega_k^T \omega_j^T \alpha_j^T S\alpha_j = \sum_{k,j}^{K} \omega_k^T \omega_j^T \left( \gamma_j^T \gamma_k \right)\]

\[= \text{Tr} \left( \sum_{k,j}^{K} \left( \omega_k^T \omega_j^T \gamma_j^T \gamma_k \right) \right)\]

\[= \text{Tr} \left( \sum_{k,j}^{K} \gamma_k \omega_k^T \omega_j^T \gamma_j^T \right)\]

\[= \| \sum_{k}^{K} \omega_k \gamma_k^T \|^2_{\text{Tr}} \text{ avec } \| A \|_{\text{Tr}} = \sqrt{\text{Trace}(A^T A)}\]

\[= \| \sum_{k}^{K} \gamma_k \omega_k^T \|^2_{\text{Tr}}\]

Then we have:

\[\arg \min_{K, \theta_{i+1}} \frac{1}{n} g(\alpha, \omega) = \arg \min_{K, \alpha = S^{-1/2} \gamma, \omega} \| S^{-1/2} N - \sum_{k=1}^{K} \gamma_k \omega_k^T \|_{\text{Tr}}^2\]

Then the solution is given by the paper [Eckart & Young (1936)] choosing $K = \text{rank}(S^{-1/2}N)$ and $\sum_{k=1}^{K} \gamma_k \omega_k^T = \sum_{r=1}^{R} \lambda_r v_r e_r^T$. Choosing $K = R$ is the best option. We now consider the matrix $M$. The minimization gives also the following properties at the optimum:

\[\text{for } k \neq j \quad \langle \gamma_k \omega_k^T, \gamma_j \omega_j^T \rangle_{\text{Tr}} = 0\]

\[\| S^{-1/2} N - \sum_{k=1}^{K} \gamma_k \omega_k^T \|_{\text{Tr}}^2 = \sum_{r=K+1}^{R} \lambda_r^2\]

\[= \| S^{-1/2} N \|^2_{\text{Tr}} - \| \sum_{k=1}^{K} \gamma_k \omega_k^T \|^2_{\text{Tr}}\]
Furthermore:

\[
\frac{1}{n} ||V^{l+1}_{\text{goal proj}} - V(\hat{\theta}_{t+l+1}^{K*})||^2_{\text{Tr}} = \frac{1}{n} ||V^{l+1}_{\text{goal proj}}||^2_{\text{Tr}} + ||(S^{1/2})^{-1} N - \sum_k \gamma_k \omega_k^T||^2 - ||(S^{1/2})^{-1} N||^2_{\text{Tr}} \\
= \sum_{r=K+1}^R \lambda_r^2 + \frac{1}{n} ||V^{l+1}_{\text{goal proj}}||^2_{\text{Tr}} - ||(S^{1/2})^{-1} N||^2_{\text{Tr}} \\
= - \sum_{r=1}^K \lambda_r^2 + \frac{1}{n} ||V^{l+1}_{\text{goal proj}}||^2_{\text{Tr}}
\]

Before looking at the impact on the Loss let’s prove the corollary 3.2. We are looking for the matrix \( \delta W_l \) minimizing the last equality. Consider the span of matrix \( S = \{ \delta W_l B_l | \delta W_l \text{matrix} \} \) and \( P \) the matrix projection on span \( S \):

\[
\arg \min_{D \in S, \theta} ||V^{l+1}_{\text{goal}} - D - V^{l+1}(\theta)||^2_{\text{Tr}} = \arg \min_{D \in S} ||PV^{l+1}_{\text{goal}} - D||^2_{\text{Tr}} \\
+ \arg \min_{\theta} \|(I - P)(V^{l+1}_{\text{goal}} - V(\theta))\|^2_{\text{Tr}}
\]

Then \( \delta W_{l+1}^* B_l \) at proposition 3.1 minimize the first norm and is the orthogonal projection of \( V^{l+1}_{\text{goal}} \) on span \( S \).

\[
(Id - P)V^{l+1}_{\text{goal}} = V^{l+1}_{\text{goal}} - \delta W_{l+1}^* B_l
\]

We also have that:

\[
\left< V^{l+1}(\delta W_{l+1}^*), V^{l+1}(\hat{\theta}_{t+l+1}^{K*}) \right>_{\text{Tr}} = 0
\]

And by definition of the orthogonal projection on linear span:

\[
\left< V^{l+1}(\hat{\theta}_{t+l+1}^{K*}), V^{l+1}_{\text{goal proj}} \right>_{\text{Tr}} = \left< V^{l+1}(\hat{\theta}_{t+l+1}^{K*}), V^{l+1}_{\text{goal proj}}(\delta W_{l+1}^*) \right>_{\text{Tr}}
\]

We note \( V^{l+1}_{\text{goal proj}}(\delta W_{l+1}^*) := V^{l+1}_{\text{goal proj}} \). The impact on the global loss is:

\[
\frac{1}{n} \sum_{i=1}^n L(f_\theta(\gamma_i, \delta W_{t+1}^{l+1}(\hat{\theta}_{t+l+1}^{K*}), \alpha_{t+1}(\gamma_i)), y_i) \\
\frac{1}{n} \sum_{i=1}^n \left( L(f_\theta(x_i), y_i) + \sigma_i(0) \left< \nabla \alpha_{t+1}(\gamma_i), L(f_\theta(x_i), y_i), v^{l+1}(\gamma_i, \hat{\theta}_{t+l+1}^{K*}, \gamma_0 \delta W_{l+1}^*) \right> \right) \\
+ o(\max(\gamma, \gamma_0))
\]

\[
\frac{1}{n} \sum_{i=1}^n L(f_\theta(x_i), y_i) - \gamma \frac{1}{n} \sigma_i(0) \left< V^{l+1}(\hat{\theta}_{t+l+1}^{K*}, V^{l+1}_{\text{goal proj}}), \delta W_{l+1}^* \right>_{\text{Tr}} \frac{1}{n} \sigma_i(0) \left< V^{l+1}(\hat{\theta}_{t+l+1}^{K*}, V^{l+1}_{\text{goal proj}}), \delta W_{l+1}^* \right>_{\text{Tr}} \\
+ o(\max(\gamma, \gamma_0))
\]

\[
\frac{1}{n} \sum_{i=1}^n L(f_\theta(x_i), y_i) - \gamma \frac{1}{n} \sigma_i(0) \left< V^{l+1}(\hat{\theta}_{t+l+1}^{K*}, V^{l+1}_{\text{goal proj}}), \delta W_{l+1}^* \right>_{\text{Tr}} - \gamma \frac{1}{n} \sigma_i(0) \left< V^{l+1}(\hat{\theta}_{t+l+1}^{K*}, V^{l+1}_{\text{goal proj}}), \delta W_{l+1}^* \right>_{\text{Tr}} \\
+ o(\max(\gamma, \gamma_0))
\]
We also have the following property:

\[
\arg \min_{\delta_{l+1}^{K}} \left\{ \frac{1}{n} \left| \left| V_{\text{goal proj}}^{l+1} - V^{l+1}(\hat{\delta}_{l+1}^{K}) \right| \right|_{2}^{2} \right\} = \arg \min_{H \geq 0} \left\{ \frac{1}{n} \left| \left| V_{\text{goal proj}}^{l+1} - V^{l+1}(\hat{\delta}_{l+1}^{K}, \delta W_{l+1}^{*}) \right| \right|_{2}^{2} \right\}
\]

Then the norm minimizing the first argmin is given by:

\[
\begin{align*}
&\arg \min_{H \geq 0} \arg \min_{\delta_{l+i+1}^{K}} \left\{ \frac{1}{n} \left| \left| V_{\text{goal proj}}^{l+1} - V^{l+1}(\hat{\delta}_{l+i+1}^{K}) \right| \right|_{2}^{2} \right\} \\
= &\arg \min_{H \geq 0} \left\{ \frac{2}{n} \left( V_{\text{goal proj}}^{l+1}, V^{l+1}(\hat{\delta}_{l+i+1}^{K}) \right)_{\text{Tr}} + \frac{1}{n} \left| \left| V^{l+1}(\hat{\delta}_{l+i+1}^{K}) \right| \right|_{2}^{2} \right\} \\
= &\arg \min_{H \geq 0} \left\{ \frac{2}{n} \left( V_{\text{goal proj}}^{l+1}, V^{l+1}(\hat{\delta}_{l+i+1}^{K}) \right)_{\text{Tr}} + \frac{1}{n} H^{2} \right\} \\
= &\arg \min_{H \geq 0} \left\{ \frac{2}{n} \left( V_{\text{goal proj}}^{l+1}, V^{l+1}(\hat{\delta}_{l+i+1}^{K}) \right)_{\text{Tr}} + \frac{1}{n} H^{2} \right\}
\end{align*}
\]

With \(V^{l+1}(\hat{\delta}_{l+i+1}^{K})\) the solution of the second arg min (ie for \(H = 1\)),

Then the norm minimizing the first argmin is given by:

\[
H^{*} = \left( V_{\text{goal proj}}^{l+1}, V^{l+1}(\hat{\delta}_{l+i+1}^{K}) \right)_{\text{Tr}} - 0
\]

Furthermore

\[
\begin{align*}
\min_{\hat{\delta}_{l+i+1}^{K}} \left\{ \frac{1}{n} \left| \left| V_{\text{goal proj}}^{l+1} - V^{l+1}(\hat{\delta}_{l+i+1}^{K}) \right| \right|_{2}^{2} \right\} &= - \sum_{r=1}^{K} \lambda_{r}^{2} + \frac{1}{n} \left| \left| V_{\text{goal proj}}^{l+1} \right| \right|_{2}^{2} \\
\min_{\hat{\delta}_{l+i+1}^{K}} \left\{ \frac{1}{n} \left| \left| V_{\text{goal proj}}^{l+1} - V^{l+1}(\hat{\delta}_{l+i+1}^{K}) \right| \right|_{2}^{2} \right\} &= \frac{1}{n} H^{2} + \frac{1}{n} \left| \left| V_{\text{goal proj}}^{l+1} \right| \right|_{2}^{2}
\end{align*}
\]

\[
\Rightarrow H^{*} = \left( V_{\text{goal proj}}^{l+1}, V^{l+1}(\hat{\delta}_{l+i+1}^{K}) \right)_{\text{Tr}} = \left( \sum_{r=1}^{K} \lambda_{r}^{2} \times \sqrt{n} \right)
\]

\[
V^{l+1}(\hat{\delta}_{l+i+1}^{K}) = H^{*} V^{l+1}(\hat{\delta}_{l+i+1}^{K})^{*}
\]

\[
\left( V^{l+1}(\hat{\delta}_{l+i+1}^{K}), V_{\text{goal proj}}^{l+1} \right)_{\text{Tr}} = H^{*} \times \left( V_{\text{goal proj}}^{l+1}, V^{l+1}(\hat{\delta}_{l+i+1}^{K})^{*} \right)_{\text{Tr}} = H^{*2}
\]

where the last equality is given by the optimisation of \(||S^{-1/2}N - \sum_{k=1}^{K} u_{k} \omega_{k}^{T}||_{\text{Tr}}^{2}\). So minimizing the scalar product \(-\left( V_{\text{goal proj}}^{l+1}(\delta W^{*}_{l+1}), V^{l+1}(\hat{\delta}_{l+i+1}^{K})^{*} \right)_{\text{Tr}}\) setting the norm of \(V^{l+1}(\hat{\delta}_{l+i+1}^{K})\) is equivalent to minimizing the norm \(||V_{\text{goal proj}}^{l+1}(\delta W^{*}_{l+1}) - V^{l+1}(\hat{\delta}_{l+i+1}^{K})^{*}||_{2}^{2} \).

\[
\frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(f_{0}(\gamma_{i}, W_{i+1}^{*}))(\alpha_{i}^{l+1, \gamma_{i}}, \omega_{i}^{l+1} (\gamma_{i}))(x_{i}, y_{i})
\]

\[
= \frac{1}{n} \sum_{i=1}^{n} L(f_{0}(x_{i}, y_{i}), \gamma_{i}) \gamma_{i} - \frac{\gamma_{i}^{2}}{n} \sum_{r=1}^{K} \lambda_{r}^{2} - \frac{\gamma_{i}}{n} \sum_{r=1}^{K} \alpha_{r}(0) \left( V^{l+1}(W_{l+1}^{*}), V_{\text{goal proj}}^{l+1} \right)_{\text{Tr}} + o(max(\gamma, \gamma_{0}))
\]

B.3 CHOLESKY DECOMPOSITION FOR S POSITIVE SEMI-DEFINITE

When matrix \(S\) is not positive definite, the following trick can be apply. Consider \(S = U \Sigma V^{T}\) the svd of \(S\). As \(S\) is symmetric \(U = V^{T}\). Perform the QR-decomposition of matrix \(\sqrt{\Sigma} U^{T} = QR\). \(Q\) an orthogonal matrix and \(R\) an upper triangular matrix. Defining \(P^{+}\) the pseudo inverse of \(P\), one can remark that \(R^{T} R = U \Sigma (Q^{-1})^{T} Q^{-1} U \sqrt{\Sigma} U^{T}\). As \(Q\) is orthogonal \(R^{T} R = U \Sigma U^{T} = U \Sigma V^{T} = S\).
B.4 Proposition B.3

Suppose $S$ is semi definite, we note $S = S^{1/2} S^{1/2T}$. Solving (7) is equivalent to taking $\omega_k = N^T \alpha_k$ and find the K first eigenvectors $\omega_k$ associated to the K largest eigenvalues $\lambda$ of the generalized eigenvalue problem:

$$NN^T \alpha_k = \lambda S \alpha_k$$

**Proof** The LOBPCG problem is equivalent to maximise the generalized Rayleigh quotient which is:

$$\alpha^* = \max_{\alpha} \frac{\alpha^T N N^T \alpha}{\alpha^T S \alpha}$$

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

$$\alpha^* = S^{1/2-1} p^*$$

And considering the SVD of $S^{1/2-1} N = \sum_R \lambda_j u_j v_j T$, then $S^{1/2-1} N N^T S^{1/2-1} = \sum_R \lambda_j u_j u_j^T$ because $j \neq i \Rightarrow u_i^T u_j = 0$ and $v_i^T v_j = 0$. So maximise the first formula is equivalent to $p_k^* = u_k$, then $\alpha_k = S^{1/2-1} u_k$. And $N^T \alpha_k = N^T S^{1/2-1} u_k = \lambda_k v_k$

We prove second corollary B.2 by induction. For $m = m' = 1$:

$$a_{i+1}(x)^{t+1} = a_{i+1}(x)^t + V(\hat{\theta}_{i+1}^{t+1}, x) \gamma + o(\gamma)$$

$$v_{\text{goal}}^{t+1, i+1}(x) = v_{\text{goal}}^{t+1, i}(x) + \nabla_{a_{i+1}(x)} L(f_\theta(x), y)^T v(\hat{\theta}_{i+1}^{t+1}, x) \gamma + o(\gamma)$$

Adding the second neuron we obtain the minimization problem:

$$\arg \min_{\alpha_2, \omega_2} ||V_{\text{goal}}^{t+1, i+1} - V^{t+1, i}(\alpha_2, \omega_2)||_{TV} + o(1)$$

B.5 Section Theory behind Greedy Growth with Proofs

One might wonder whether a greedy approach on layer growth might get stuck in a non-optimal state. 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:

$$\inf_{f} \sum_{x \in \mathcal{D}} ||f(x) - f^*(x)||^2$$

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

**Proposition B.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 $\omega$, for any desired precision $\gamma$, and for any activation function $\sigma$ provided it is not a polynomial, then there exists a neural network $\tilde{g}$ with one hidden layer (possibly quite large when $\gamma$ is small) and with this activation function $\sigma$, such that

$$\forall x, ||g(x) - \tilde{g}(x)|| \leq \gamma$$

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 $\sigma$, 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 \( \gamma = \frac{1}{100} \| f^* - f \|_{L^2}, \) where \( \langle a \mid b \rangle_{L^2} = \frac{1}{|x|} \int x \in \omega a(x) b(x) \, dx. \) This way we obtain a one-hidden-layer neural network \( g \) with activation function \( \sigma \) such that:

\[
\forall x \in \omega, \quad -\gamma \leq g(x) - g^*(x) \leq \gamma
\]

\[
\forall x \in \omega, \quad g(x) = f^*(x) - f(x) + a(x)
\]

with \( \forall x \in \omega, \ |a(x)| \leq \gamma. \)

Then:

\[
\forall x \in \omega, \quad f^*(x) - (f(x) + g(x)) = -a(x)
\]

\[
\forall x \in \omega, \quad (f^*(x) - h(x))^2 = a^2(x)
\]

(12)

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.

\[
\| f^* - h \|^2_{L^2} = \| a \|^2_{L^2}
\]

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 \( \gamma = \frac{1}{100} \sqrt{\frac{1}{|D|} \sum_{x \in D} |f(x) - f^*(x)|^2}, \) as Equation (12) then yields:

\[
\sum_{x \in D} (f^*(x) - h(x))^2 = \sum_{x \in D} a^2(x) \leq \frac{1}{100} \sum_{x \in D} (f^*(x) - f(x))^2
\]

which proves the proposition as stated.

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

\[\square\]

**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 \( \sigma \) 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 \( \sigma \) 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.

\[\square\]

**Proposition B.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 \( L^2 \) 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 \( L^2 \) regression). That is, denoting by \( a_i \sigma(W_i \cdot x) \) these \( N \) neurons:

\[
\langle \sum_{i=1}^{N} a_i \sigma(w_i \cdot x) \mid g \rangle_{L^2} = K < 0
\]

i.e.

\[
\sum_{i=1}^{N} \langle a_i \sigma(w_i \cdot x) \mid g \rangle_{L^2} = K < 0
\]
Now, by contradiction, if there existed no neuron \( i \) among these ones such that 
\[
\langle a_i \sigma (w_i \cdot x) \mid g \rangle_{L^2} \leq \frac{1}{N} K
\]
then we would have:
\[
\forall i \in [1, N], \quad \langle a_i \sigma (w_i \cdot x) \mid g \rangle_{L^2} > \frac{1}{N} K
\]
\[
\sum_{i=1}^{N} \langle a_i \sigma (w_i \cdot x) \mid g \rangle_{L^2} > K
\]
hence a contradiction. Then necessarily at least one of the \( N \) neurons satisfies
\[
\langle a_i \sigma (w_i \cdot x) \mid g \rangle_{L^2} \leq \frac{1}{N} K < 0
\]
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, our greedy approach will not get stuck in a situation where one would need to add many neurons simultaneously to decrease the loss: it is always feasible by a single neuron. One can express a lower bound on how much the loss has improved (for the best such neuron), but it is not a very good one without further assumptions on \( f \).

**Proposition B.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 consequently manipulates infinitesimal quantities. Though we perform line search in a second step and consequently add non-infinitesimal neurons, our first optimization problem relies on the linearization of the activation function by requiring the added neuron to have infinitely small input weights, without which it would be much harder to solve. This proposition confirms that such neuron does exist indeed.
C ADDITIONAL EXPERIMENTAL RESULTS

All the experiments are performed 20 times on MNIST Dataset and the models are trained with Adam($lr = 0.0001, \mu = 0$, batchsize = 100) with 7 CPU. For the classic model, neurons are initialized with Kaiming Normal. For our approach, we always start with a model of size $[1,1]$ initialized with Kaiming Normal and we expand its architecture using 3.2 every 0.05 seconds. The batch-size, $n_{mb}$, for estimating $\theta(\gamma)_{l+1}(\gamma)$ and $\delta W_{l+1}^i$ is $n_{mb} = 100$ at first step and for every layer. After applying our method on every layers, $n_{mb}$ is updated as $n_{mb}^l \leftarrow n_{mb}^l \times \sqrt{\max(\text{neurons}_{l-1}, \text{neurons}_{l+1})}$, where $\text{neurons}_l$ indicates the amount of neurons at layer $l$. The $y-axis$ is accuracy on test set.

C.1 CHANGING TRAINING TIME BETWEEN EACH ARCHITECTURE EXPANSION STEP

In this part we modify the training time between each architecture growth. We apply 8 times our method on each hidden layer, the final architecture is $[222, 71]$

![Figure 8](image.png)

Figure 8: The different shades of red correspond to different training time between each architectures growth in seconds.

C.2 SMALL ARCHITECTURE

Same plot as [6] but with others architecture.

![Figure 9](image.png)

Figure 9: Right interpolation on interval of size 0.01. Middle : accuracy as a function of mini-batches. Right : accuracy plotted as a function of the mean of time in each category : classic training and our approach.

C.3 EIGENVALUES

In this section we apply our method 15 times on each layer. After applying 8 times our method, the accuracy of the system does not increase significantly. We have that $\mathbb{E}_{x,y \sim D}[z^T v_{\text{goal}}(x)]$ for every
vector $z$, as explained in part\[5\] Looking at the eigenvalues for the first and second hidden layers and

Figure 10: Comparison between the classic training of architecture $[1000, 1000]$ and our approach of architecture $[929, 141]$. All models are trained for 20 seconds we plot here mean and standard deviation for accuracy on test set. For our method we trained the model for 0.05 seconds between each architecture growth. Right interpolation on interval of size 0.01. Middle : accuracy as a function of epochs. Right : accuracy plotted as a function of the mean of time in each category: classic training and our approach.

estimate decrease for the loss, ie $\sum_k \lambda_k^2$:

Figure 11: mean and standard deviation for $\frac{1}{2} \sum_k \lambda_k^2$ for first and second hidden layers as a function of the number of architecture growths

Figure 12: mean and standard deviation for $\max(\lambda_k)$ for first and second hidden layers as a function of the number of architecture growths
C.4 RANDOM VS OPTIMIZATION

When performing the quadratic optimization (5), we obtain the optimal direction for $(\hat{\alpha}_k^*, \hat{\omega}_k^*)_{k=1}^R$. It is also possible to generate randomly the new neurons and compute the amplitude factors. This second option have the benefit of being less time consuming, but it would project the desired direction on those random vectors and would affect the accuracy score compared to optimal solution defined in 3.1.

Figure 13: Experiment performed 20 times on the MNIST dataset: a starting model in black $[1, 1]$ is initialized according to normal Kaiming, then is duplicated to give the red and the green model. The structure of the red model is modified by our method to reach the structure $[110, 51]$ while the green model is extended with random neurons. Then all models are trained for 5 seconds. The white space for the red model corresponds to the quadratic optimisation and the computation of the amplitude factor while for the green model it corresponds only to the computation of the amplitude factor.