

TINY / FoGrow vs other one-shot Neural Architecture Search strategies

Manon Verbockhaven, Stella Douka, Theo Rudkiewicz, François P. Landes, Stéphane Rivaud, Sylvain Chevallier, Guillaume Charpiat
`prenom.nom@inria.fr`

TAU team, LISN, INRIA Saclay, Université Paris-Saclay

February 2025

Plan

- One-shot techniques in neural architecture search (NAS) :
 - ▶ NAS optimization problem
 - ▶ one shot methods
- FoGrow vs two others one shot techniques :
 - ▶ FoGrow a gradient-based and first order approximation method
 - ▶ GradMax (how to train faster a network ?)
 - ▶ NORTH (how to expand the functional space of the network ?)
 - ▶ comparison on CIFAR100
- Move away from first order approximation
 - ▶ amplitude factor
 - ▶ theoretical advantages
 - ▶ experiments

Plan : Part I

- One shot techniques in NAS :
 - ▶ NAS optimization problem
 - ▶ one shot techniques

NAS : the optimization problem

Consider a task $\mathcal{T}$ and a corresponding dataset $\mathcal{X}$. Let $\mathcal{A}$ a set of architecture and $\mathcal{L}$ a loss function, we would like to solve :

$$\arg \min_{a(\cdot) \in \mathcal{A}, \mathbf{w} \in \mathbb{R}^{d(a)}} \mathcal{L}_{train}(a(\mathbf{w}))$$



$$\iff \arg \min_{a \in \mathcal{A}} \mathcal{L}_{train}(a(\mathbf{w}^*)) \quad s.t. \quad \mathbf{w}^* := \arg \min_{\mathbf{w}} \mathcal{L}_{train}(a(\mathbf{w}))$$

NAS Optimization problem

$$\arg \min_{a \in \mathcal{A}} \mathcal{L}_{val}(a(\mathbf{w}^*)) \quad s.t. \quad \mathbf{w}^* := \arg \min_{\mathbf{w}} \mathcal{L}_{train}(a(\mathbf{w})) \quad (1)$$

NAS : a two-steps optimization procedure

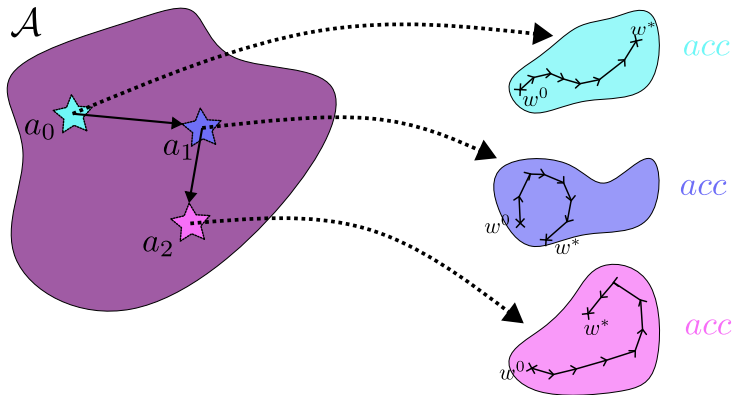


Figure 1: Optimization of a network architecture with the decoupling between the search into the space of architecture $\mathcal{A}$ and the space of parameters $\mathbb{R}^{d(a)}$.

One-Shot methods

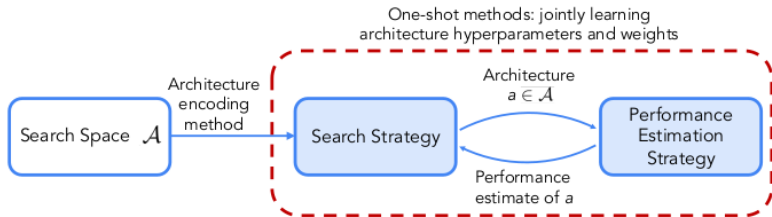


Figure 2: Graph of operation of NAS methods from [1]. A set of possible architectures $\mathcal{A}$ is chosen. A starting architecture a is drawn from $\mathcal{A}$ to encode a neural network $\mathcal{N}$. The performance of $\mathcal{N}$ is then evaluated and is processed by the search strategy, which might draw another architecture a from $\mathcal{A}$.

categories of one-shot methods ([1], [10])

With an *hypernetwork*

A hypernetwork is a network that generates the weights of another network ([2]).

The strategies with a supernet

A supernet is a large network that contains all the architectures of the search space $\mathcal{A}$. Each architecture of $\mathcal{A}$ is a sub-graph of the supernet. The strategy consists of selecting the appropriate sub-graph of the supernet ([3], [4]).

gradient-based

A gradient-based method searches in $\mathcal{A}$ using the back-products of the standard backpropagation (gradient estimation method to update network parameters) ([5]–[9]).

Convolutional Neural Fabrics : a supernetwork

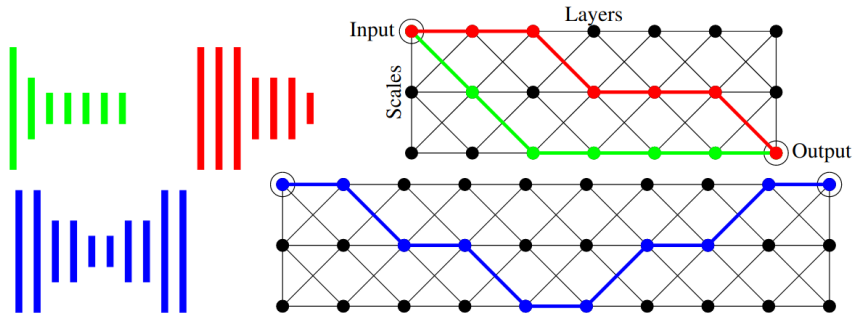


Figure 3: Supernet from [3].

Plan : Part II

- FoGrow vs two other one shot techniques :
 - ▶ FoGrow a gradient-based and first order approximation method
 - ▶ GradMax (how to train faster a network ?)
 - ▶ NORTH (how to expand the functional space of the network ?)
 - ▶ comparison on CIFAR100

Notations

$$\mathcal{D} := \{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^N, \quad \mathcal{A}, \quad f_{\mathcal{A}(\theta)} := f_\theta, \quad \mathcal{L} : \mathbb{R}^d \mapsto \mathbb{R}^+$$

$$\mathbf{b}_0 = \mathbf{x}, \quad \mathbf{b}_L = f_\theta$$

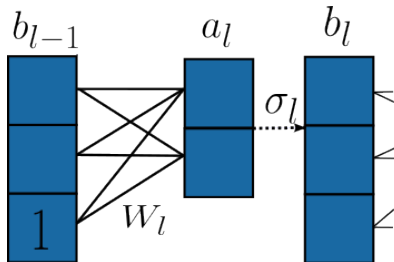


Figure 4: Notations.

FoGrow : a gradient-based method

FoGrow

$$\mathcal{T}_{\mathcal{A}}^{\mathbf{a}_l} := \left\{ \frac{\partial \mathbf{a}_l}{\partial \theta} \delta \theta \mid \text{s.t. } \delta \theta \in \Theta_{\mathcal{A}} \right\}$$

$$\mathbf{v}_{\nabla}^l := -\nabla_{\mathbf{a}_l} \mathcal{L}(f_{\theta})$$

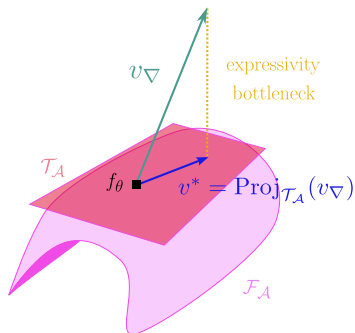


Figure 5: Expressivity bottleneck.

$$\begin{aligned} \Psi^l &:= \min_{\mathbf{v}^l \in \mathcal{T}_{\mathcal{A}}^{\mathbf{a}_l}} \mathbb{E}_{\mathbf{x} \sim \mathcal{D}} \left[\|\mathbf{v}^l(\mathbf{x}) - \mathbf{v}_{\nabla}^l(\mathbf{x})\|^2 \right] \\ &= \min_{\delta \theta} \mathbb{E}_{\mathbf{x} \sim \mathcal{D}} \left[\|\mathbf{v}^l(\mathbf{x}, \delta \theta) - \mathbf{v}_{\nabla}^l(\mathbf{x})\|^2 \right] \end{aligned}$$

FoGrow : a gradient-based method

FoGrow

$$\mathcal{T}_{\mathcal{A}}^{a_l} := \left\{ \frac{\partial \mathbf{a}_l}{\partial \theta} \delta \theta \mid \text{s.t. } \delta \theta \in \Theta_{\mathcal{A}} \right\}$$

$$\mathbf{v}_{\nabla}^l := -\nabla_{\mathbf{a}_l} \mathcal{L}(f_{\theta})$$

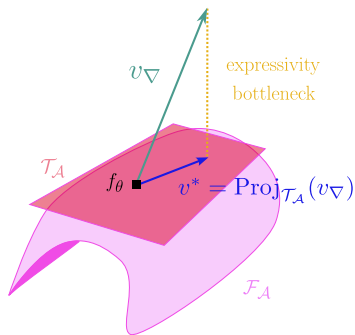


Figure 5: Expressivity bottleneck.

$$\begin{aligned} \Psi^l &:= \min_{\mathbf{v}^l \in \mathcal{T}_{\mathcal{A}}^{a_l}} \sum_{i=1}^n \|\mathbf{v}^l(\mathbf{x}_i) - \mathbf{v}_{\nabla}^l(\mathbf{x}_i)\|^2 \\ &= \min_{\delta \theta} \frac{1}{n} \|\mathbf{V}^l(\mathbf{X}, \delta \theta) - \mathbf{V}_{\nabla}^l(\mathbf{X})\|_{\text{Tr}}^2 \end{aligned}$$

Updating the current architecture

$$\Psi^l := \min_{\mathbf{v}^l \in \mathcal{T}_{\mathcal{A}^l}} \sum_{i=1}^n \left\| \mathbf{v}^l(\mathbf{x}_i) - \mathbf{v}_{\nabla}^l(\mathbf{x}_i) \right\|^2$$

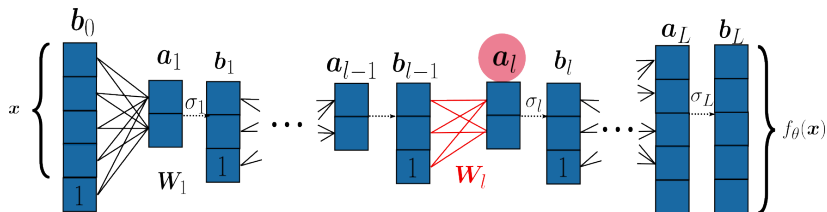


Figure 6: Approximating the expressivity bottleneck by modifying the parameters at layer l .

Adding neurons

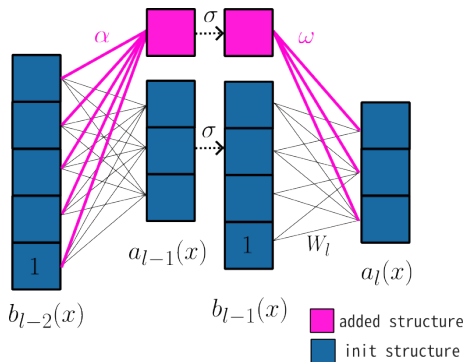


Figure 7: Adding one neuron to layer l in cyan, with connections in cyan. Here, $\alpha \in \mathbb{R}^5$ and $\omega \in \mathbb{R}^3$.

The optimization problems

$$\delta \mathbf{W}^* = \arg \min_{\delta \mathbf{W}} \frac{1}{n} \|\mathbf{V}(\delta \mathbf{W}) - \mathbf{V}_{\nabla}\|_{\text{Tr}}^2 \quad (2)$$

$$\mathbf{A}^*, \mathbf{\Omega}^* = \arg \min_{K, \text{rk}(\mathbf{\Omega} \mathbf{A}^T) = K} \frac{1}{n} \|\mathbf{V}((\mathbf{A}, \mathbf{\Omega})) - \mathbf{V}_{\perp}\|_{\text{Tr}}^2 \quad (3)$$

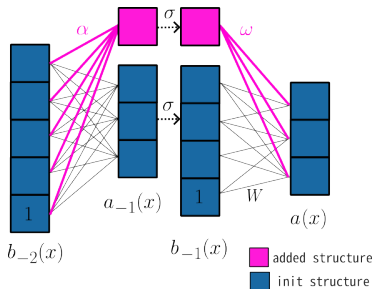


Figure 8: Notations

The optimization problems

$$\delta \mathbf{W}^* = \arg \min_{\delta \mathbf{W}} \frac{1}{n} \|\delta \mathbf{W} \mathbf{B}_{-1} - \mathbf{V}_{\nabla}\|_{\text{Tr}}^2 \quad (2)$$

$$\mathbf{A}^*, \mathbf{\Omega}^* = \arg \min_{K, \text{rk}(\mathbf{\Omega} \mathbf{A}^T) = K} \frac{1}{n} \|\mathbf{\Omega} \mathbf{A}^T \mathbf{B}_{-2} - \mathbf{V}_{\perp}\|_{\text{Tr}}^2 \quad (3)$$

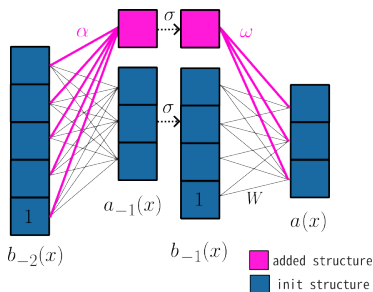


Figure 8: Notations

The optimization problems

$$\delta \mathbf{W}^* = \arg \min_{\delta \mathbf{W}} \frac{1}{n} \|\delta \mathbf{W} \mathbf{B}_{-1} - \mathbf{V}_{\nabla}\|_{\text{Tr}}^2 \quad (2)$$

$$\mathbf{A}^*, \mathbf{\Omega}^* = \arg \min_{K, \text{rk}(\mathbf{\Omega} \mathbf{A}^T) = K} \frac{1}{n} \|\mathbf{\Omega} \mathbf{A}^T \mathbf{B}_{-2} - \mathbf{V}_{\perp}\|_{\text{Tr}}^2 \quad (3)$$

Best Update and new neurons

Considering a batch of size n to construct the post activation and the desired update matrices, then for any depth l and $z \in \{1, 2\}$, noting $\mathbf{B}_{-z} := \mathbf{B}_{l-z}$, $\mathbf{V}_{\nabla} := \mathbf{V}_{\nabla}^l$, taking $K = \text{rank}(\mathbf{V}_{\perp} \mathbf{B}_{-2}^T)$, then :

$$\delta \mathbf{W}^* = \mathbf{V}_{\nabla} \mathbf{B}_{-1}^T (\mathbf{B}_{-1} \mathbf{B}_{-1}^T)^+ \quad (4)$$

$$\mathbf{\Omega}^* \mathbf{A}^{*T} = \mathbf{V}_{\perp} \mathbf{B}_{-2}^T (\mathbf{B}_{-2} \mathbf{B}_{-2}^T)^+ \quad (5)$$

Best update and new neurons formulas

Best Update and new neurons

Considering a batch of size n to construct the post activation and the desired update matrices, then for any depth l and $z \in \{1, 2\}$, noting $B_{-z} := B_{l-z}$, $V_{\nabla} := V_{\nabla}^l$, ... and $S_{-z} := \frac{1}{n} B_{-z} B_{-z}^T$, then

$$\delta W^* = V_{\nabla} B_{-1}^T (B_{-1} B_{-1}^T)^+ \quad (6)$$

$$\left(\Omega^* A^{*T} \right)_{|K} = \left(V_{\perp} B_{-2}^T (B_{-2} B_{-2}^T)^{-\frac{1}{2}} \right)_{|K} S_{-2}^{-\frac{1}{2}} \quad (7)$$

Best update vs new neurons formulas

Best Update and new neurons

Considering a batch of size n to construct the post activation and the desired update matrices, then for any depth l and $z \in \{1, 2\}$, noting $B_{-z} := B_{l-z}$, $V_{\nabla} := V_{\nabla}^l$, ... and $S_{-z} := \frac{1}{n} B_{-z} B_{-z}^T$, taking $K = \text{rank}(V_{\nabla \text{proj}} B_{-2}^T)$, then :

$$\begin{aligned} \delta W^* &= V_{\nabla} B_{-1}^T (B_{-1} B_{-1}^T)^+ \\ \Omega^* A^{*T} &= V_{\perp} B_{-2}^T (B_{-2} B_{-2}^T)^{-\frac{1}{2}} \quad \begin{matrix} = \frac{1}{n} B_{-2} B_{-2}^T \\ S_{-2}^{-\frac{1}{2}} \end{matrix} \end{aligned}$$

Best update and new neurons formulas

Best Update and new neurons

Considering a batch of size n to construct the post activation and the desired update matrices, then for any depth l and $z \in \{1, 2\}$, noting $B_{-z} := B_{l-z}$, $V_{\nabla} := V_{\nabla}^l$, taking $K = \text{rank}(V_{\nabla \text{proj}} B_{-2}^T)$, then :

$$\delta W^* = V_{\nabla} B_{-1}^T (B_{-1} B_{-1}^T)^+ \quad (8)$$

$$\Omega^* A^{*T} = V_{\perp} B_{-2}^T (B_{-2} B_{-2}^T)^+ \quad (9)$$

Similar new neurons initialization methods

GradMax ([5]) & NORTH ([6])

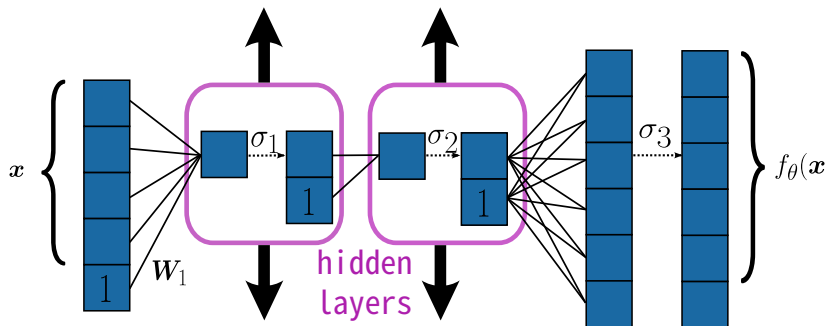


Figure 8: Thin feed forward network with two hidden layers.

GradMax & NORTH

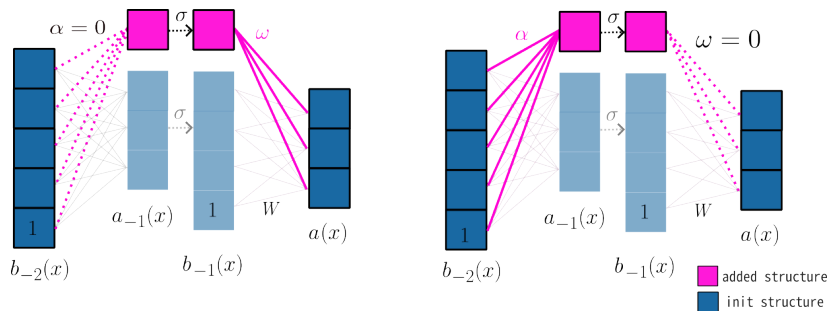


Figure 9: left : GradMax initialization; right : NORTH initialization

GradMax initialization philosophy

objective : Accelerate the training of a network.

Updating the network parameters with $\delta\theta$:

$$\mathcal{L}^{t+1} = \mathcal{L}^t + \langle \nabla_{\theta} \mathcal{L}, \delta\theta \rangle + o(\|\delta\theta\|)$$

Choosing the gradient direction, i.e. $\delta\theta = -\eta \nabla_{\theta} \mathcal{L}$:

$$\mathcal{L}^{t+1} = \mathcal{L}^t - \eta \|\nabla_{\theta} \mathcal{L}\|^2 + o(\eta)$$

Idea : Maximizing the gradient according to the new parameters.

GradMax trick

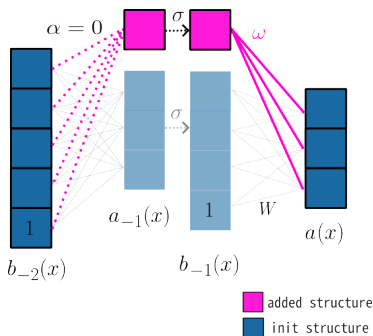


Figure 10: GradMax new neurons initialization.

$$\mathcal{L}^{t+1} \approx \mathcal{L}^t + \langle \nabla_{\theta} \mathcal{L}, \delta \theta \rangle + \langle \nabla_{\Omega} \mathcal{L}, \overbrace{\delta \Omega}^{-\nabla_{\Omega} \mathcal{L}} \rangle + \langle \nabla_{\mathbf{A}} \mathcal{L}, \overbrace{\delta \mathbf{A}}^{-\nabla_{\mathbf{A}} \mathcal{L}} \rangle$$

$$\mathcal{L}^{t+1} \approx \mathcal{L}^t - \|\nabla_{\theta} \mathcal{L}\|^2 - \left(\|\nabla_{\Omega} \mathcal{L}\|^2 + \overbrace{\|\nabla_{\mathbf{A}} \mathcal{L}\|^2}^{\text{quadratic in } \Omega} \right)$$

GradMax initialization

GradMax initialization :

$$\Omega^* := \arg \max_{\Omega} \|\nabla_{\mathbf{A}} \mathcal{L}\|^2 \quad s.t. \quad \|\Omega\|^2 \leq c \quad (10)$$

$$= \arg \max_{\Omega} \left\| \overbrace{B_{-2} \mathbf{V}_{\nabla}^T}^{:=N} \Omega \right\|^2 \quad s.t. \quad \|\Omega\|^2 \leq c \quad (11)$$

GradMax initialization

GradMax initialization :

$$\mathbf{\Omega}^* := \arg \max_{\mathbf{\Omega}} \|\mathbf{N}\mathbf{\Omega}\|^2 \quad s.t. \quad \|\mathbf{\Omega}\|^2 \leq c \quad (10)$$

GradMax initialization

GradMax initialization :

$$\Omega^* := \arg \max_{\Omega} \|\mathbf{N}\Omega\|^2 \quad s.t. \quad \|\Omega\|^2 \leq c \quad (10)$$

FoGrow initialization :

$$\mathbf{A}^* \Omega^{*T} = \overbrace{(\mathbf{B}_{-2} \mathbf{B}_{-2}^T)}^S + \overbrace{\mathbf{B}_{-2} \mathbf{V}_{\perp}^T}^{:= \mathbf{N}_{\perp}}$$
$$\exists \tilde{c} \quad \Omega^* := \arg \max_{\Omega} \left\| \mathbf{S}^{-\frac{1}{2}} \mathbf{N}_{\perp} \Omega \right\|^2 \quad s.t. \quad \|\Omega\| \leq \tilde{c}$$

GradMax initialization

GradMax initialization :

$$\mathbf{\Omega}^* := \arg \max_{\mathbf{\Omega}} \|\mathbf{N}\mathbf{\Omega}\|^2 \quad s.t. \quad \|\mathbf{\Omega}\|^2 \leq c \quad (10)$$

FoGrow initialization :

$$\exists \tilde{c} \quad \mathbf{\Omega}^* := \arg \max_{\mathbf{\Omega}} \left\| \mathbf{S}^{-\frac{1}{2}} \mathbf{N}_{\perp} \mathbf{\Omega} \right\|^2 \quad s.t. \quad \|\mathbf{\Omega}\| \leq \tilde{c} \quad (11)$$

From FoGrow to GradMax

First order development of the loss :

$$\begin{aligned}\mathcal{L}(f_{\theta \oplus (\mathbf{A}, \Omega)}) &:= \frac{1}{n} \sum_i \ell(f_{\theta}(\mathbf{x}_i)) \\ &= \mathcal{L}(f_{\theta}) - \frac{1}{n} \langle \mathbf{V}, \mathbf{V}_{\nabla} \rangle_{\text{Tr}} + o\left(\frac{1}{n} \|\mathbf{V}\|\right)\end{aligned}$$

The expressivity bottleneck :

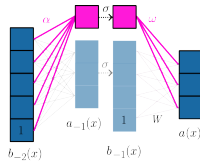
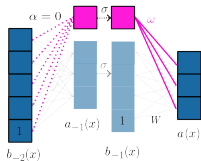
$$\Psi = \min_{\delta \theta} \frac{1}{n} \|\mathbf{V}(\mathbf{X}, (\mathbf{A}, \Omega)) - \mathbf{V}_{\perp}(\mathbf{X})\|_{\text{Tr}}^2$$

Proposition

$$\forall \mathbf{D} \in \mathbb{R}^{p,q}, \mathbf{B} \in \mathbb{R}^{k,q},$$

$$\exists \tilde{c} \in \mathbb{R} \quad \text{s.t.,} \quad \arg \min_H \|\mathbf{D} - \mathbf{H}\mathbf{B}\|^2 = \arg \max_{H, \|\mathbf{H}\mathbf{B}\|^2 \leq \tilde{c}} \langle \mathbf{D}, \mathbf{H}\mathbf{B} \rangle$$

GradMax vs FoGrow



GradMax	FoGrow
$\arg \max_{\Omega} \ \nabla_{\mathbf{A}} \mathcal{L}(f_{\theta})\ ^2 \text{ s.t. } \ \Omega\ \leq c$	$\arg \max_{\mathbf{A}, \Omega} \langle \mathbf{A} \Omega^T, \mathbf{N}_{\perp} \rangle \text{ s.t. } \ \Omega \mathbf{A}^T \mathbf{B}\ ^2 \leq c$
$\Downarrow$	$\Downarrow$
$\arg \max_{\Omega} \ \mathbf{N}_{\nabla} \Omega\ ^2 \text{ s.t. } \ \Omega\ ^2 \leq c$	$\arg \max_{\Omega} \ \mathbf{S}^{-\frac{1}{2}} \mathbf{N}_{\perp} \Omega\ ^2 \text{ s.t. } \ \Omega\ \leq \tilde{c}$

Table 1: Optimization problems defining the fan-out weights of the new neurons for GradMax method (left) and FoGrow method (right).

GradMax & NORTH

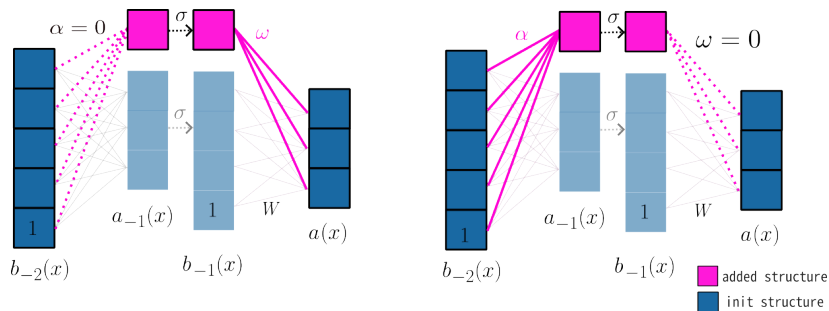
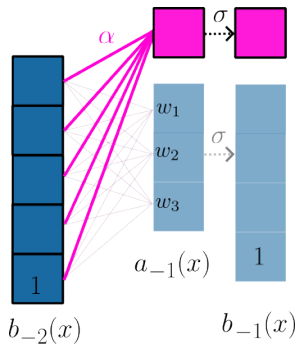


Figure 11: left : GradMax initialization; right : NORTH initialization

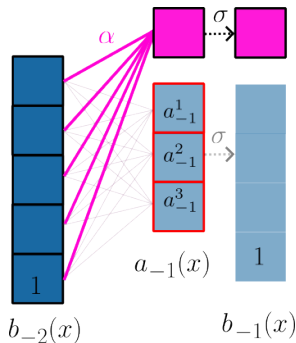
NORTH (1): Weight-based



$$\alpha \in \{w_1, w_2, w_3\}^\perp$$

Figure 12: Notation NORTH, zoom.

NORTH (2) :activation-based



$$\mathbf{v}(\cdot, \boldsymbol{\alpha}) \in \{\mathbf{a}_{-1}^1(\cdot), \mathbf{a}_{-1}^2(\cdot), \mathbf{a}_{-1}^3(\cdot)\}^\perp$$

Figure 13: Notation NORTH, zoom.

$$\boldsymbol{\alpha}^{\text{act}} = \mathbf{S}^+ \mathbf{B}_{l-2} \mathbf{V}_{\text{Ker}(\mathbf{A}_{l-1})} \mathbf{r} \quad \text{with} \quad \mathbf{r} \sim \mathcal{N}(0, I) \quad (12)$$

NORTH vs FoGrow

NORTH	FoGrow
$\mathbf{S}^+ \mathbf{B}_{l-2} \mathbf{V}_{\text{Ker}(\mathbf{A}_{l-1})} \mathbf{r}, \quad \mathbf{r} \sim \mathcal{N}(0, \mathbf{I})$	$\mathbf{S}^+ \mathbf{B}_{l-2} \overbrace{\mathcal{P}_{\text{Ker}(\mathbf{B}_{l-1})} \mathbf{V}_{\nabla}^T}^{\mathbf{v}_{\perp}^T}[:, i]$

Table 2: Fan-in weights initialization for NORTH method (left) and FoGrow method (right).

Conclusion : Initialization differences

Input weights $\mathbf{A}$:

NORTH	FoGrow
$\mathbf{S}^+ \mathbf{B}_{l-2} \mathbf{V}_{\text{Ker}(\mathbf{A}_{l-1})} \mathbf{r}, \quad \mathbf{r} \sim \mathcal{N}(0, \mathbf{I})$	$\mathbf{S}^+ \mathbf{B}_{l-2} \mathcal{P}_{\text{Ker}(\mathbf{B}_{l-1})} \mathbf{V}_{\nabla}^T[:, i]$

Table 3: Fan-in weights initialization for NORTH method (left) and FoGrow method (right).

Output weights Ω :

GradMax	FoGrow
$\arg \max_{\Omega} \ \mathbf{N}_{\nabla} \Omega\ ^2 \text{ s.t. } \ \Omega\ ^2 \leq c$	$\arg \max_{\Omega} \ \mathbf{S}^{-\frac{1}{2}} \mathbf{N}_{\perp} \Omega\ ^2 \text{ s.t. } \ \Omega\ \leq \tilde{c}$

Table 4: Optimization problems defining the fan-out weights of the new neurons for GradMax method (left) and FoGrow method (right).

Plan : Part III

- Move away from first order approximation
 - ▶ amplitude factor
 - ▶ theoretical advantages
 - ▶ experiments

The amplitude factor (go faster !)

The amplitude factor γ

A new neuron with input and output weights α, ω is multiplied by the amplitude factor γ with the operation $\gamma(\alpha, \omega)$ as :

$$(\alpha, \omega) \leftarrow (\sqrt{\gamma}\alpha, \sqrt{\gamma}\omega). \quad (13)$$

For an input $\mathbf{x}$, then :

$$\mathbf{v}(\mathbf{x}, (\alpha, \omega)) = \sqrt{\gamma}\omega\sigma(\sqrt{\gamma}\alpha^T \mathbf{b}_{-2}(\mathbf{x})).$$

And for an input $\mathbf{x}$, if $\|\gamma\| \ll 1$, then :

$$\mathbf{v}(\mathbf{x}, (\alpha, \omega)) = O(\|\gamma\|).$$

The best amplitude factor (go faster !)

The best amplitude factor γ^*

For a batch $\mathbf{X}_\gamma$, we can compute the best amplitude factor γ^* as :

$$\gamma^* := \arg \min_{\gamma \in [0, u]} \sum_{\mathbf{x}_i \in \mathbf{X}_\gamma} \mathcal{L}(f_{\theta \oplus \gamma(\mathbf{A}, \Omega)})(\mathbf{x}_i), \mathbf{y}_i)$$

For large γ^* :

~~$$\mathcal{L}(f_{\theta \oplus \gamma(\mathbf{A}, \Omega)}) = \mathcal{L}(f_\theta) - \gamma^* \frac{1}{n} \langle \mathbf{V}, \mathbf{V}_\nabla \rangle_{\text{Tr}} + o(\gamma^*)$$~~

GradMax & NORTH : no impact

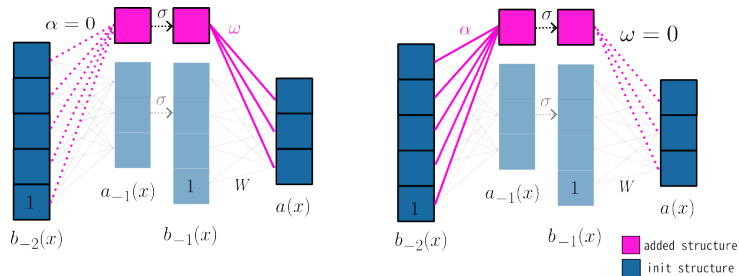


Figure 14: right : GradMax initialization; left : NORTH initialization

The amplitude factor for the strategy

Where to first add neurons ?

Using first order information, i.e. $\gamma \ll 1$:

$$\mathcal{L}(f_{\theta \oplus \gamma(\mathbf{A}, \Omega)}) = \mathcal{L}(f_{\theta}) - \frac{1}{n} \langle \mathbf{v}', \mathbf{v}_{\nabla}' \rangle_{\text{Tr}} + o\left(\frac{1}{n} \|\mathbf{v}\|\right) \quad (14)$$

$$l^* := \arg \max_l \langle \mathbf{v}', \mathbf{v}_{\nabla}' \rangle_{\text{Tr}} \quad (15)$$

The amplitude factor for the strategy

Where to first add neurons ?

Using first order information, i.e. $\gamma \ll 1$:

$$\mathcal{L}(f_{\theta \oplus \gamma(\mathbf{A}, \Omega)}) = \mathcal{L}(f_{\theta}) - \frac{1}{n} \langle \mathbf{V}^I, \mathbf{V}_{\nabla}^I \rangle_{\text{Tr}} + o\left(\frac{1}{n} \|\mathbf{V}\|\right) \quad (14)$$

$$I^* := \arg \max_I \langle \mathbf{V}^I, \mathbf{V}_{\nabla}^I \rangle_{\text{Tr}} \quad (15)$$

Consider this scalar product at layer I and $I + 1$:

$$\langle \mathbf{V}^I, \mathbf{V}_{\nabla}^I \rangle \quad \text{and} \quad \langle \mathbf{V}^{I+1}, \mathbf{V}_{\nabla}^{I+1} \rangle \quad (16)$$

$$= \langle \mathbf{K} \mathbf{V}^I, \mathbf{V}_{\nabla}^{I+1} \rangle \quad (17)$$

where $\mathbf{K}(\mathbf{x}) := \left(\frac{\partial \sigma(\mathbf{u})}{\partial \mathbf{u}} \Big|_{\mathbf{u}=\mathbf{W}\mathbf{a}_I(\mathbf{x})} \mathbf{W}_I \right)^T$.

The amplitude factor for the strategy

Where to first add neurons ?

Using first order information, i.e. $\gamma \ll 1$:

$$\mathcal{L}(f_{\theta \oplus \gamma}(\mathbf{A}, \Omega)) = \mathcal{L}(f_{\theta}) - \frac{1}{n} \langle \mathbf{V}', \mathbf{V}_{\nabla}' \rangle_{\text{Tr}} + o\left(\frac{1}{n} \|\mathbf{V}\|\right) \quad (14)$$

$$l^* := \arg \max_l \langle \mathbf{V}', \mathbf{V}_{\nabla}' \rangle_{\text{Tr}} \quad (15)$$

Using non-linear information :

$$\Delta_l := \mathcal{L}(f_{\theta \oplus \gamma^*}(\mathbf{A}, \Omega)) - \mathcal{L}(f_{\theta}) \quad (16)$$

$$l := \arg \max_l \Delta_l \quad (17)$$

Experiment : Algorithm (1)

Algorithm 1: Grow a network with the strategy s

Data: $s = \text{Strategy}$

Result: Neural Network N

Initialize a two linear layers network with one neuron per layer;

for t **in** $[1, \dots, T]$ **do**

$depth = s(t)$;

 Compute the first neuron α_1, ω_1 of at $depth$;

 Compute the amplitude factor γ^* for this new neuron;

 Multiply the neuron with its amplitude factor;

 Add it to the architecture;

 Evaluate the performances of the network;

for d **in** $[1, 2, 3]$ **do**

 Compute the best update at layer d and its amplitude factor;

 Multiply the best update with its amplitude factor;

 Update the architecture with this best update;

 Evaluate the performances of the network;

end

end

Experiment : Algorithm (2)

Algorithm 2: Select depth to grow the network

Data: s = Strategy, t = iteration

Result: $depth$ where to add a neuron

if $s = Naive$ **then**

$depth = np.mod(t, 2) + 1$;

else if $s = MaxEigenValue$ **then**

for d in $[1, 2]$ **do**

 Compute the first neuron α_1, ω_1 at d ;

 Compute the first order information $\lambda_1^d = \langle \mathbf{V}, \mathbf{V}_\perp \rangle$ at d ;

end

$depth = \arg \max_{d=1,2} \{\lambda_1^d\}$;

else if $s = AmplitudeFactor$ **then**

for d in $[1, 2]$ **do**

 Compute the first neuron α_1, ω_1 at d ;

 Compute the amplitude factor γ^* for this new neuron;

 Evaluate the decrease of loss Δ_d if the architecture is expanded with
 $(\sqrt{\gamma^*} \alpha_1, \sqrt{\gamma^*} \omega_1)$

end

$depth = \arg \max_{d=1,2} \{\Delta_d\}$;

return $depth$;

Experiment : Figure

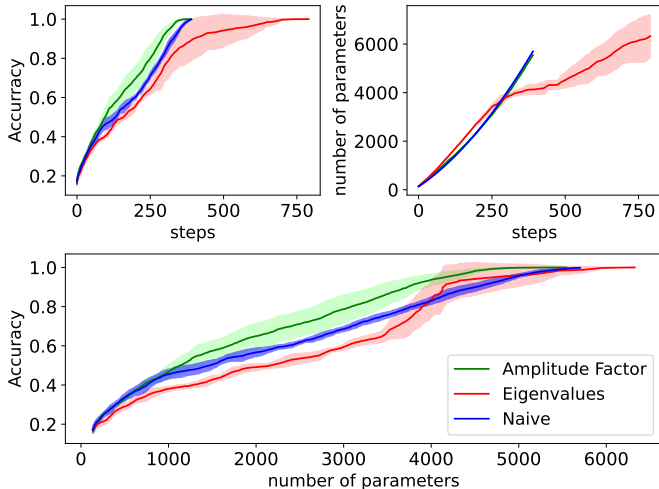
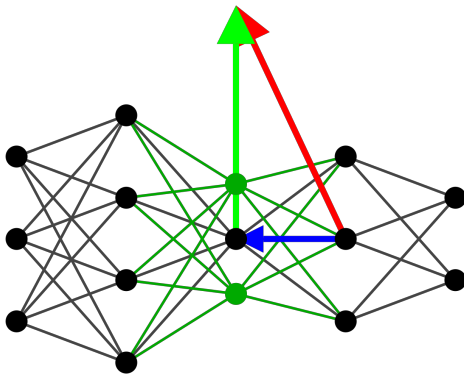


Figure 15: Accuracy and number of parameters of growing networks with different addition strategies for $\mathbf{x} \sim \mathcal{N}(0, I_{50})$ and $\mathbf{y} = \arg \max_{k=1, \dots, 10} \left\{ \sin(k \sum_{j=1}^{50} \mathbf{x}[j]) \right\}$.

Conclusion : Amplitude factor

- Accelerate training.
- Better compare the impact of the new neurons on the loss.
- More resource demanding (fast convergence ?).

Thank you



- [1] Colin White, Mahmoud Safari, Rhea Sukthanker, et al. *Neural Architecture Search: Insights from 1000 Papers*. 2023. arXiv: 2301.08727 [cs.LG]. URL: <https://arxiv.org/abs/2301.08727>.
- [2] Andrew Brock, Theodore Lim, J. M. Ritchie, et al. *SMASH: One-Shot Model Architecture Search through HyperNetworks*. arXiv:1708.05344 [cs]. Aug. 2017. DOI: 10.48550/arXiv.1708.05344. URL: <http://arxiv.org/abs/1708.05344> (visited on 01/22/2025).
- [3] Shreyas Saxena and Jakob Verbeek. “Convolutional Neural Fabrics”. In: *Advances in Neural Information Processing Systems*. Vol. 29. Curran Associates, Inc., 2016. URL: https://papers.nips.cc/paper_files/paper/2016/hash/07811dc6c422334ce36a09ff5cd6fe71-Abstract.html (visited on 01/17/2025).

- [4] 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>.
- [5] Utku Evci, Bart van Merriënboer, Thomas Unterthiner, et al. “GradMax: Growing Neural Networks using Gradient Information”. In: *International Conference on Learning Representations*. 2022. URL: https://openreview.net/forum?id=qjN4h_wwU0.
- [6] Kaitlin Maile, Emmanuel Rachelson, Hervé Luga, et al. “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=SW0g-arIg9>.
- [7] Xiaoliang Dai, Hongxu Yin, and Niraj K. Jha. “NeST: A Neural Network Synthesis Tool Based on a Grow-and-Prune Paradigm”. In: *IEEE Transactions on Computers* 68.10 (2019), pp. 1487–1497. DOI: 10.1109/TC.2019.2914438.

- [8] Lemeng Wu, Bo Liu, Peter Stone, et al. “Firefly Neural Architecture Descent: a General Approach for Growing Neural Networks”. In: *Advances in Neural Information Processing Systems*. Vol. 33. Curran Associates, Inc., 2020, pp. 22373–22383. URL: <https://proceedings.neurips.cc/paper/2020/hash/fdbe012e2e11314b96402b32c0df26b7-Abstract.html> (visited on 12/17/2024).
- [9] Manon Verbockhaven, Théo Rudkiewicz, Guillaume Charpiat, et al. “Growing Tiny Networks: Spotting Expressivity Bottlenecks and Fixing Them Optimally”. In: *Transactions on Machine Learning Research* (2024). ISSN: 2835-8856. URL: <https://openreview.net/forum?id=hbtG6s6e7r>.
- [10] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. “Neural Architecture Search: A Survey”. In: *Journal of Machine Learning Research* 20.55 (2019), pp. 1–21. URL: <http://jmlr.org/papers/v20/18-598.html>.