Neural Architecture Growth by fixing Expressivity Bottlenecks

Manon Verbockhaven, Barbara Hajdarevic,
Sylvain Chevallier, Guillaume Charpiat
prenom.nom@inria.fr




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

14 décembre 2023
JRAF, Grenoble

Neural Architecture Growth by fixing Expressivity Bottlenecks

Manon Verbockhaven, Barbara Hajdarevic,
Sylvain Chevallier, Guillaume Charpiat
and Stella Douka

and you!

Post-doc/SRP position available: contact me!

`prenom.nom@inria.fr`






?

Overview

- Introduction & Neural Architecture Search
- Expressivity bottlenecks
- Best neurons to add
- Experimental results on fixed architecture
- Extension to DAG
- Discussion

I - Introduction

Introduction

Computational ressources required by Deep Learning:

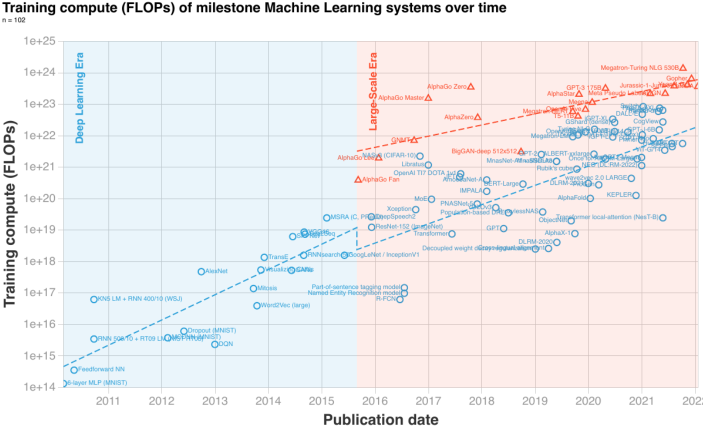


Figure 3: Trends in training compute of $n102$ milestone ML systems between 2010 and 2022. Notice the emergence of a possible new trend of large-scale models around 2016. The trend in the remaining models stays the same before and after 2016.

Compute Trends Across Three Eras of Machine Learning, Sevilla et al., IJCNN 2022

Introduction

Computational ressources required by Deep Learning:

- larger and larger models (ex: GPT)
- larger because more powerful in practice
  (cf scaling laws, provided more data is available)
- more and more powerful $\implies$ more and more used
  (and this is just the beginning)

Environmental impact:

- training cost: impressive for LLM, yet quite small w.r.t. usage in industry (cf M. Jay's presentation: 20% at FB)
- carbon impact: debatably less than when done by Humans
- other environmental impacts (full life cycle, incl. hardware and data): ?

$\implies$ here, computational complexity

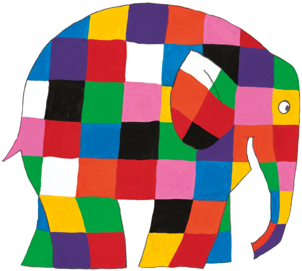
(pros: hardware independent; cons: without memory access, cf yesterday talks + Anais Boumendil's ongoing PhD thesis)

Introduction

Common paradigm:
train large architectures

Pros:

- approximate any function *(universal approximation theorems)*
- nice optimization properties: gradient descent leads to good minima *(many possible optimization directions)*
- scaling laws *(better results with more data)*

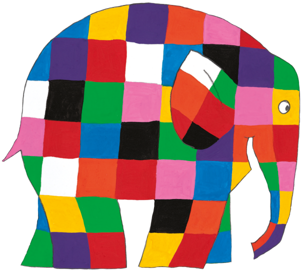


Cons:

- it's heavy (to train and to apply)
- need for reduction techniques afterwards *(pruning, quantization, tensorization... or distillation)*

Introduction

Common paradigm:
train large architectures



Pros:

- approximate any function *(universal approximation theorems)*
- nice optimization properties: gradient descent leads to good minima *(many possible optimization directions)*
- scaling laws *(better results with more data)*

Cons:

- it's heavy (to train and to apply)
- need for reduction techniques afterwards *(pruning, quantization, tensorization... or distillation)*

$\Longrightarrow$ Finding the right architecture directly by optimizing it ?

NAS : Neural Architecture Search

Single vs. multi-task:

- single task, from scratch (no prior knowledge)
- multi-task learning, transfer, meta DL (sharing information between tasks)

Architecture search:

- by hand
- exploration (fancy random search by trial & error): genetic algorithms[1], reinforcement learning[2]
  $\Longrightarrow$ sensitive to exploration hyper-parameters, needs a lot of computational resources.
- gradient-based methods
  - pruning large networks: DARTS[3]
  - growing small networks: GradMax[4]

1 : Compositional pattern producing networks: A novel abstraction of development, K. Stanley, 2007
2 : Neural Architecture Search with Reinforcement Learning, B. Zoph et al, ICLR 2017
3 : DARTS: Differentiable Architecture Search, H. Liu, K. Simonyan & Y. Yang, ICLR 2019
4 : GradMax: Growing Neural Networks using Gradient Information, U. Evci et al, ICLR 2022

NAS : Neural Architecture Search

Single vs. multi-task:

- single task, from scratch (no prior knowledge)
- multi-task learning, transfer, meta DL (sharing information between tasks)

Architecture search:

- by hand
- exploration (fancy random search by trial & error): genetic algorithms[1], reinforcement learning[2]
  $\Longrightarrow$ sensitive to exploration hyper-parameters, needs a lot of computational resources.
- gradient-based methods
  - pruning large networks: DARTS[3]
  - growing small networks: GradMax[4]

1 : Compositional pattern producing networks: A novel abstraction of development, K. Stanley, 2007
2 : Neural Architecture Search with Reinforcement Learning, B. Zoph et al, ICLR 2017
3 : DARTS: Differentiable Architecture Search, H. Liu, K. Simonyan & Y. Yang, ICLR 2019
4 : GradMax: Growing Neural Networks using Gradient Information, U. Evci et al, ICLR 2022

Starting with a small neural network

Training a small network:

- faster learning, less memory
- the solution found by gradient descent is poor

$\Longrightarrow$ adapt the architecture *during* training:

- estimate and localize potential **expressivity bottlenecks**, and fix them *on the fly*, without trial/error.

II - Expressivity Bottlenecks

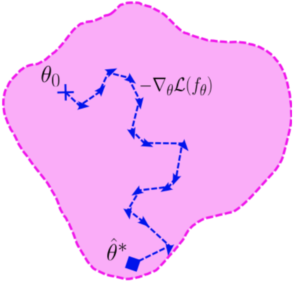
Definitions, Goals, Objectives

For a given neural network with architecture $\mathcal{A}$:

- What are *expressivity bottlenecks* ?
  - Where are they ?
  - How to quantify them ?
  - in a computationally efficient manner ?

Optimizing Neural Networks

Gradient descent in the space of parameters θ: converges to a local optimum



$$|\mathcal{L}(\hat{\theta}^*) - \mathcal{L}(\theta^*)| \to 0$$

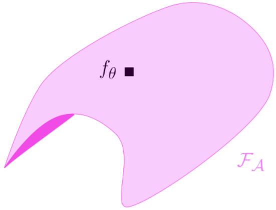
Notations

- Dataset $\mathcal{D} := \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^N \in \left(\mathbb{R}^p \times \mathbb{R}^d\right)^N iid \sim \mathcal{P}$
- Neural Network $f_\theta : \mathbb{R}^p \to \mathbb{R}^d$
- Loss function $\mathcal{L} : \left(\mathbb{R}^d\right)^2 \to \mathbb{R}^+$

Functional geometry

Mathematical objects

- Architecture space : $\Theta_{\mathcal{A}}$
- $\mathcal{F}_{\mathcal{A}} = \{f_\theta \mid \theta \in \Theta_{\mathcal{A}}\}$

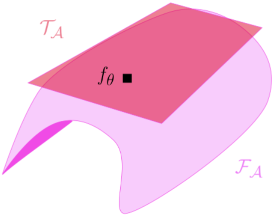


Functional geometry

Mathematical objects

- Architecture space : $\Theta_{\mathcal{A}}$
- $\mathcal{F}_{\mathcal{A}} = \{f_\theta \mid \theta \in \Theta_{\mathcal{A}}\}$
- Tangent space at f_θ: $\mathcal{T}_{\mathcal{A}}^{f_\theta} := \mathcal{T}_{\mathcal{A}} =$

$$\left\{ \frac{\partial f_\theta}{\partial \theta} \delta\theta \;\middle|\; \text{s.t. } \delta\theta \in \Theta \right\}$$

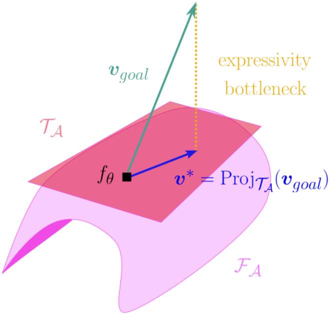


$$g \in \mathcal{T}_{\mathcal{A}} \iff \exists \delta\theta \;\; s.t. \;\; g(\boldsymbol{x}) = f_\theta(\boldsymbol{x}) + \frac{\partial f(\boldsymbol{x})}{\partial \theta} \delta\theta$$

Functional geometry

Mathematical objects

- Architecture space : $\Theta_{\mathcal{A}}$
- $\mathcal{F}_{\mathcal{A}} = \{f_\theta \mid \theta \in \Theta_{\mathcal{A}}\}$
- Tangent space at f_θ:
  $\mathcal{T}_{\mathcal{A}}^{f_\theta} := \mathcal{T}_{\mathcal{A}} =$
  $$\left\{ \frac{\partial f_\theta}{\partial \theta}\, \delta\theta \;\middle|\; \text{s.t. } \delta\theta \in \Theta \right\}$$



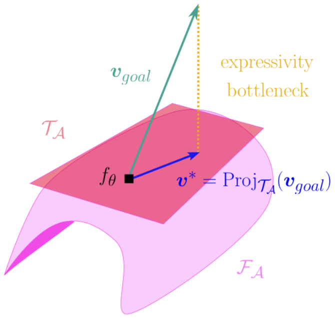
$$\boldsymbol{v}_{\text{goal}} := -\nabla_f \mathcal{L}(f)\big|_{f=f_\theta}$$

$\boldsymbol{v}^*$: best possible move within $\mathcal{T}_{\mathcal{A}}$

Functional geometry

Mathematical objects

- Architecture space : $\Theta_{\mathcal{A}}$
- $\mathcal{F}_{\mathcal{A}} = \{f_\theta \mid \theta \in \Theta_{\mathcal{A}}\}$
- Tangent space at f_θ:
  $\mathcal{T}_{\mathcal{A}}^{f_\theta} := \mathcal{T}_{\mathcal{A}} =$
  $$\left\{ \frac{\partial f_\theta}{\partial \theta}\, \delta\theta \;\middle|\; \text{s.t. } \delta\theta \in \Theta \right\}$$



$\boldsymbol{v}_{\text{goal}} := -\nabla_f \mathcal{L}(f)\big|_{f=f_\theta}$

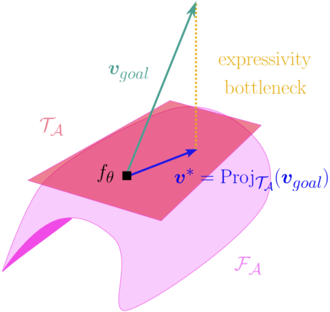
$\boldsymbol{v}^*$: best possible move within $\mathcal{T}_{\mathcal{A}}$ $= -\nabla_\theta \mathcal{L}(f_\theta)$

$\Longrightarrow$ Problem solved! Easy!

Functional geometry

Mathematical objects

- Architecture space : $\Theta_{\mathcal{A}}$
- $\mathcal{F}_{\mathcal{A}} = \{f_\theta \mid \theta \in \Theta_{\mathcal{A}}\}$
- Tangent space at f_θ:
  $\mathcal{T}_{\mathcal{A}}^{f_\theta} := \mathcal{T}_{\mathcal{A}} =$
  $$\left\{ \frac{\partial f_\theta}{\partial \theta} \delta\theta \;\middle|\; \text{s.t. } \delta\theta \in \Theta \right\}$$



$\boldsymbol{v}_{\text{goal}} := -\nabla_f \mathcal{L}(f)\big|_{f=f_\theta}$

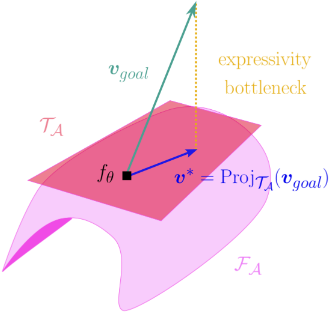
$\boldsymbol{v}^*$: best possible move within $\mathcal{T}_{\mathcal{A}} = -\frac{\partial f_\theta}{\partial \theta} \nabla_\theta \mathcal{L}(f_\theta)$

$\Longrightarrow$ Problem solved! Easy!

Functional geometry

Mathematical objects

- Architecture space : $\Theta_{\mathcal{A}}$
- $\mathcal{F}_{\mathcal{A}} = \{f_\theta \mid \theta \in \Theta_{\mathcal{A}}\}$
- Tangent space at f_θ:
  $\mathcal{T}_{\mathcal{A}}^{f_\theta} := \mathcal{T}_{\mathcal{A}} =$
  $$\left\{ \frac{\partial f_\theta}{\partial \theta}\, \delta\theta \;\middle|\; \text{s.t. } \delta\theta \in \Theta \right\}$$



$\boldsymbol{v}_{\text{goal}} := -\nabla_f \mathcal{L}(f)|_{f=f_\theta}$

$\boldsymbol{v}^*$: best possible move within $\mathcal{T}_{\mathcal{A}}$ $= -\left(\frac{\partial f_\theta}{\partial \theta} \frac{\partial f_\theta}{\partial \theta}^T\right)^+ \frac{\partial f_\theta}{\partial \theta} \nabla_\theta \mathcal{L}(f_\theta)$

$\Longrightarrow$ Problem solved! ~~Easy!~~ "natural" gradient

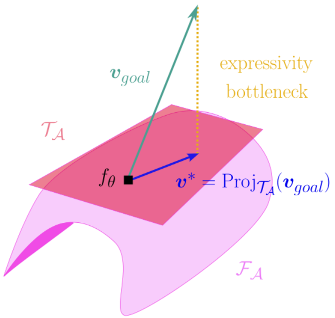
Functional geometry

Expressivity Bottleneck

$$\arg\min_{\boldsymbol{v}\in\mathcal{T}_\mathcal{A}} \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim\mathcal{P}}\left[\left|\left|\boldsymbol{v}_{\text{goal}}(\boldsymbol{x}) - \boldsymbol{v}(\boldsymbol{x})\right|\right|^2\right]$$

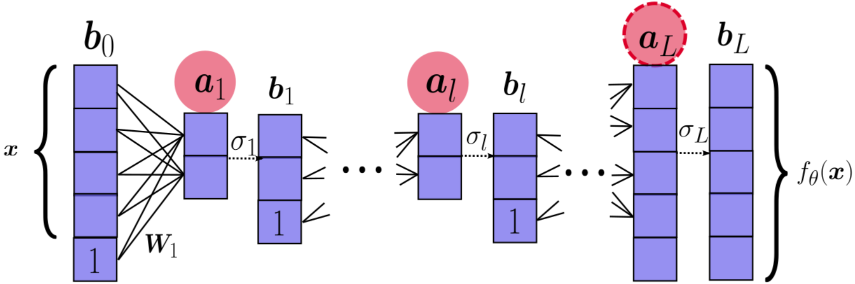
$$= \arg\min_{\boldsymbol{v}\in\mathcal{T}_\mathcal{A}} \left\{ D_f\mathcal{L}(f)(\boldsymbol{v}) + \frac{1}{2}\|\boldsymbol{v}\|^2 \right\}$$

Best move $\boldsymbol{v}^* =$ projection indeed



Generalizing to each layer: $\boldsymbol{v}$ and $\boldsymbol{v}_{\text{goal}}$ at layer l

Notations: a_l, b_l: pre- and post-activations at layer l



Definition ($\boldsymbol{v}_{\text{goal}}^l$ desired update)

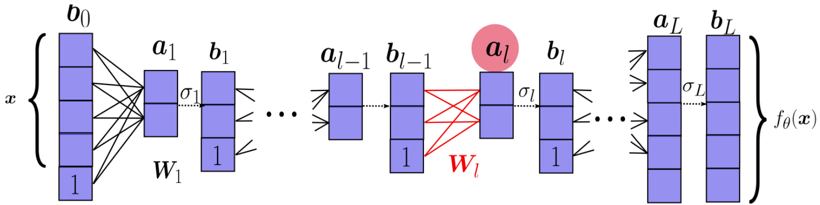
$$\boldsymbol{v}_{\text{goal}}{}^l(\boldsymbol{x}) := -\eta \nabla_{\boldsymbol{a}_l(\boldsymbol{x})} \mathcal{L}(f_\theta(\boldsymbol{x}), \boldsymbol{y})$$

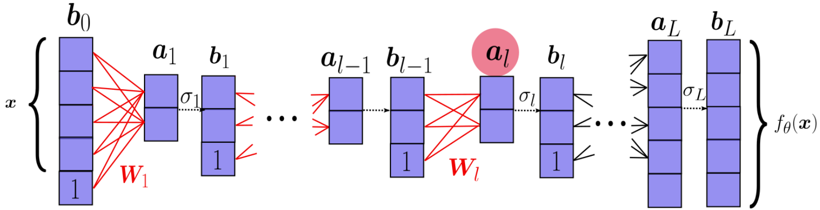
obtained by back-propagation

Definition ($\boldsymbol{v}^l$ possible update)

$$\boldsymbol{v}^l(\boldsymbol{x}, \delta\theta) := \frac{\partial \boldsymbol{a}_l(\boldsymbol{x})}{\partial \theta} \delta\theta$$

Simplification: non-convex problem w.r.t. all parameters, to convex problem w.r.t. layer params





Convex optimization

Best update at layer l

Linear regression from layer input $\boldsymbol{b}_{l-1}(\boldsymbol{x})$ to desired output variation $\boldsymbol{v}_{\text{goal}}^{l}(\boldsymbol{x})$:

$$\delta \boldsymbol{W}_l^* := \arg\min_{\delta\theta} ||\boldsymbol{V}_{\text{goal}}^l - \delta \boldsymbol{W}_l \, \boldsymbol{B}_{l-1}||^2$$

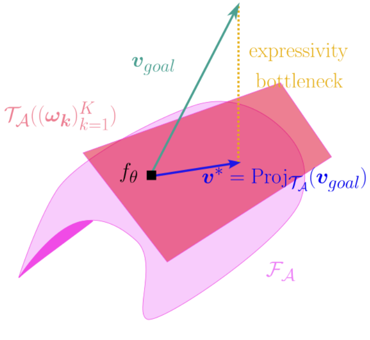
$$\delta \boldsymbol{W}_l^* = \frac{1}{n} \boldsymbol{V}_{\text{goal}}^l \boldsymbol{B}_{l-1}^T \left(\frac{1}{n} \boldsymbol{B}_{l-1} \boldsymbol{B}_{l-1}^T\right)^{-1}$$

with $\boldsymbol{B}_{l-1} := \begin{pmatrix} \boldsymbol{b}_{l-1}(\boldsymbol{x}_1) & \dots & \boldsymbol{b}_{l-1}(\boldsymbol{x}_n) \end{pmatrix}$

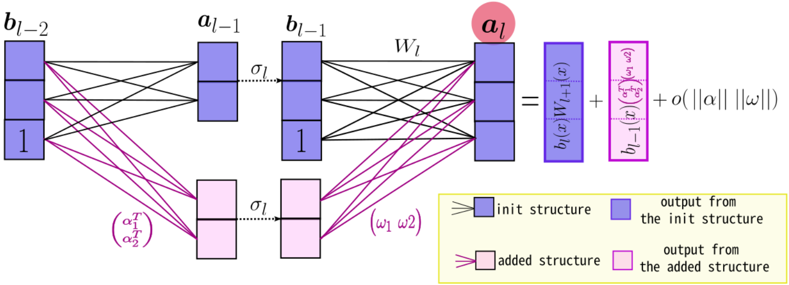
$\Longrightarrow$ can now quantify and locate expressivity bottlenecks

III - Best neurons to add

Augmenting tangent space by adding neurons



Architecture change to fix expressivity bottleneck



Best neurons to reduce expressivity bottleneck at layer l

$$\arg\min_{A,\Omega} \overbrace{\left|\left| V^l_{\text{goal}} - \mathbf{V}^{l*} - \boldsymbol{\Omega} \boldsymbol{A}^T \boldsymbol{B}_{l-2} \right|\right|^2_{\text{Tr}}}^{\text{Expressivity Bottleneck}}$$

with neuron weights $\boldsymbol{\Omega} := \begin{pmatrix}\boldsymbol{\omega}_1 & \dots & \boldsymbol{\omega}_K\end{pmatrix}$ and $\boldsymbol{A} := \begin{pmatrix}\boldsymbol{\alpha}_1 & \dots & \boldsymbol{\alpha}_K\end{pmatrix}$

Solution: by SVD

Overall algorithm

For each layer l:

- layer expressivity bottleneck: $||\mathbf{V}^l_{\text{goal}} - \mathbf{V}^{l^*}||$
- with $\mathbf{V}^{l^*}$: best parameter move with current architecture (by SVD)
- estimate best neurons to add to layer l (by SVD)

Then:

- grow most promising layer (with a line search on added neuron weights)
- update all layers with $\mathbf{V}^{l^*}$ (+ line search) or gradient descent

Overall algorithm

For each layer l:

- layer expressivity bottleneck: $||\boldsymbol{V}^l_{\text{goal}} - \boldsymbol{V}^{l^*}||$
- with $\boldsymbol{V}^{l^*}$: best parameter move with current architecture (by SVD)
- estimate best neurons to add to layer l (by SVD)

Then:

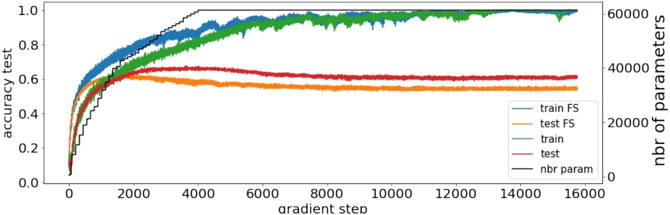
- grow most promising layer (with a line search on added neuron weights)
- update all layers with $\boldsymbol{V}^{l^*}$ (+ line search) or gradient descent

Computational cost:

- SVD: cubic, but paradoxically negligible
- line search: less negligible
- estimation of matrices with sufficiently many samples: even less negligible $\implies$ statistical significance: accuracy $\simeq O(\frac{1}{\sqrt{N}})$
$\implies N \propto (Width \times FilterSize)^2 / \# Pixels$

IV - Experimental results

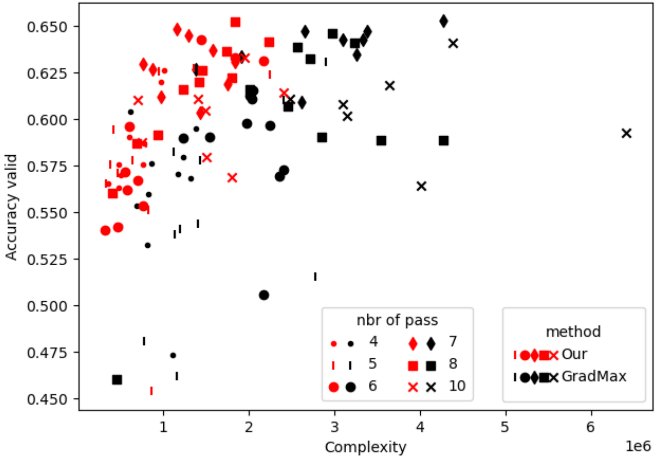
CIFAR-10



Accuracy as a function of gradient step

- succeeds in total overfitting (100% on train)
- similar learning curve as the standard approach with all neurons from the beginning (FS = final structure found by our method and reinitialized, retrained from scratch)
- without need for choosing in advance the number of neurons/layer

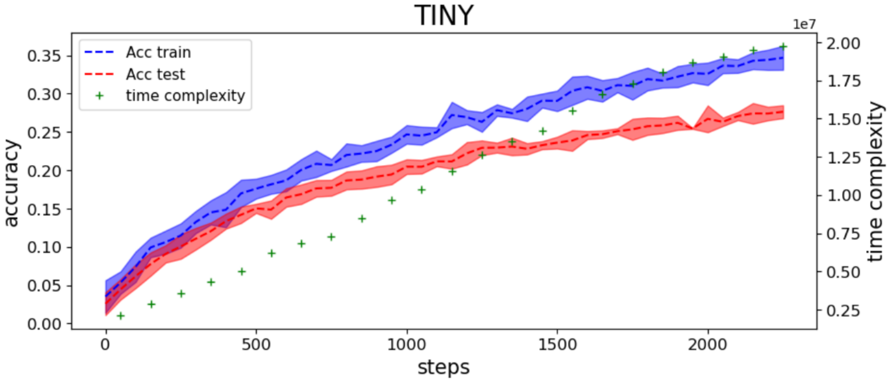
CIFAR-10



Accuracy as a function of complexity at test time

- better Pareto front

CIFAR-100

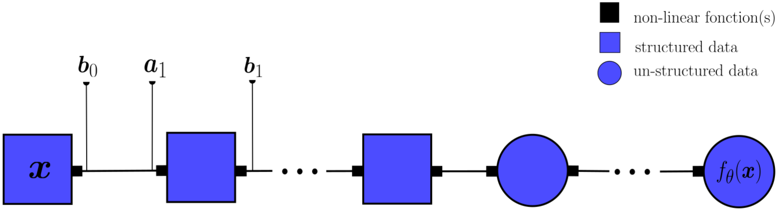


ResNet-18 on CIFAR-100

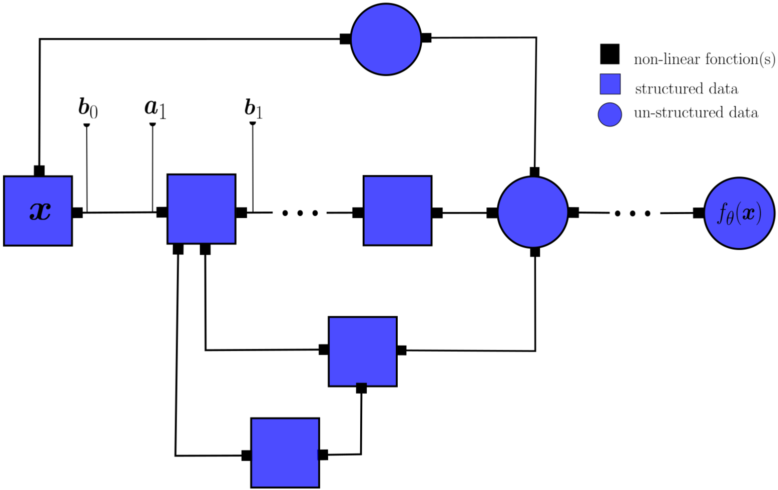
V - Adding layers

Extension to graph growth

Extension to addition of layers (any DAG): work in progress



Extension to graph growth



Extension to graph growth

How to add a new layer on the fly?

- adding a new layer = adding neurons to an empty layer
- same approach

though first-order approximations of activation functions are not sufficient anymore when adding a layer parallel to a linear layer

- same quadratic problem, but with different terms inside
- or use random tries, random projections (but in a principled manner)
- or perform gradient descent on the new neuron parameters

VI - Conclusion

Discussion

- Greedy approach: provably not an issue
  Proposition:
  There always exists a neuron to add that can improve the loss
- Avoids redundancy (at each time step)
  but final number of neurons might be non-optimal (for given target accuracy)
- Addition strategy (based on size/performance compromise)
  $\Longrightarrow$ compare loss gain to computational complexity increase:

$$\delta\mathcal{L} \quad vs. \quad AddedComplexity$$

  $\Longrightarrow$ same as considering $\mathcal{L}' = \mathcal{L} + \alpha\, Complexity$
- Reasonable runtime: similar to a single standard training
  (one run to be compared with NAS: many random tries of architectures)
- Other architectures: convolution = done, attention = to do

Discussion (bis)

Challenges

- Overfit ?
- Spurious correlations (when estimating best neurons to add)
  $\Longrightarrow$ random matrix theory to estimate eigenvalue significance
  $\Longrightarrow$ quantify required dataset size for reliable neuron estimation
  but guaranties are gone if one uses gradient descent afterwardss

  Data-hungry? Data augmentation? Complexity? Better/other estimators?
- Optimization issue if using gradient descent: different learning rates
- Based on linear correlations between inputs and desired output variations of a layer
  $\Longrightarrow$ if stuck, consider higher-order or wise mix with combinatorics / random tries

Thanks!

Thanks for your attention!

Preprint:
https://www.lri.fr/~gcharpia/Expressivity_bottlenecks_preprint.pdf

Reminder: we are searching for a post-doc/Starting-Research-Position!