Growing neural networks in the context of a Transformer architecture

Internship report by Léo Burgund

Master 1: Mathematics and Artificial intelligence

Université Paris-Saclay, Laboratoire interdisciplinaire des sciences du numérique (Equipe TAU), Institut national de recherche en informatique et en automatique

Gif-sur-Yvette, 2025-08-29

supervised by
Guillaume Charpiat
Sylvain Chevallier
Stéphane Rivaud

Acknowledgements

I would like to thank the INRIA / LISN laboratory and the TAU team for their support during this internship. I am particularly grateful to Stéphane Rivaud for their supervision and helpful discussions.

Contents

List of Figures

List of Algorithms

1 Introduction

1.1 Motivation and problem

Neural networks have become the de facto universal function approximators across modern computing, underpinning systems for perception, language, and decision-making (LeCun et al., 2015; Silver et al., 2016). Their empirical success, however, comes with steep training and inference costs that strain memory, energy, and latency budgets, especially at the scale of today's foundation models (Brown et al., 2020; Touvron et al., 2023). These models are often intentionally over-parameterized to ease optimization and improve transfer (Allen-Zhu et al., 2019; Arora et al., 2018), but their size then necessitates aggressive post-training compression (quantization, pruning, and distillation) before deployment (Han et al., 2016; Hinton et al., 2015). While this post-hoc slimming can be effective, it implicitly amounts to a form of neural architecture search (NAS): we try to recover a smaller network that preserves the input–output behavior of the original behemoth (Frankle & Carbin, 2019). Unfortunately, classical NAS is itself computationally intensive because each candidate architecture typically requires end-to-end training and evaluation (Real et al., 2017; Zoph & Le, 2017). This motivates frugal, training-aware alternatives that can steer architecture capacity precisely where the task demands it, without incurring the full cost of exploring many models from scratch (Chen et al., 2016; Cortes et al., 2017; Maile et al., 2022).

A recent line of work takes exactly this perspective: instead of searching over architectures externally, it grows the network during training by spotting expressivity bottlenecks and adding the most beneficial neurons in closed form. Concretely, TAU team's paper (Verbockhaven et al., 2024) tracks a functional gradient signal to locate layers where the current model underfits, and then insert weights that maximize validation-loss improvement under a constrained, one-step approximation. Their derivations cover affine (linear + bias) mappings and extend to convolutions, yielding formulas for optimal neuron additions that can be computed from statistics accumulated online, so the architecture adapts as learning progresses.

The internship builds directly on TAU team's growth-based view and tackles the core architectural component of modern Transformers: self-attention. Unlike a single affine layer, dot-product attention computes logits as $\frac{QK^\top}{\sqrt{k}}$, where $Q = XW_Q$ and $K = XW_K$. Algebraically, this introduces second-order interactions: the logits contain products of pairs of input coordinates (via $X(.)X^\top$) and are bilinear in the parameters (W_Q, W_K) through the coupled term $W_Q W_K^\top$ (Vaswani et al., 2017). After softmax, the output further multiplies by $V = XW_V$. In short, attention is not just "another linear map"; it is a composition whose core scoring operator behaves quadratically in the inputs and bilinearly in the parameters. As a result, the closed-form neuron-addition rules derived for affine/convolutional layers in (Verbockhaven et al., 2024) paper cannot be ported verbatim: the key separability that enables independent, per-neuron optimization in linear layers breaks down because query and key parameters co-determine the same interaction term.

From a methodological standpoint, this raises two technical challenges. First, feature construction: in affine layers the sufficient statistics for "best new neuron" live in first-order feature covariances; in attention, we must reason about pairwise token-wise interactions and their projections through W_Q and W_K. Second, parameter coupling: whereas linear layers admit decoupled updates for incoming and outgoing weights of a new unit, attention ties the effectiveness of a query update to a matching key update (and potentially value/projection parameters), turning the selection problem into a jointly constrained optimization. Our approach therefore

develops growth criteria and closed-form (or closed-form-like) joint updates for attention heads that approximate the optimal change in the functional objective while remaining inexpensive enough to compute from accumulated batch statistics. The goal is to recover the same spirit as the affine case—spot the bottleneck, add the most useful capacity there—but in a setting where the operator's algebra is fundamentally second-order.

Practically, frugal growth within attention is well-motivated by deployment realities. Foundation-model pipelines often rely on attention-heavy backbones whose inference cost scales with sequence length and head dimensions; adding capacity only where the functional gradient indicates under-fitting can yield better accuracy–efficiency trade-offs than uniform widening (Michel et al., 2019). Moreover, such growth can complement standard compression: instead of first training a large model and then shrinking it, we seek to shape the model to the task while it learns, reducing the need for brute-force over-parameterization upfront.

Empirically, we validate the derived attention-growth formulas on small-scale image benchmarks—CIFAR-10/100 and Imagenette—to check correctness and implementation soundness. We caution that these datasets are not ideal for showcasing Transformer benefits: studies note that vanilla Vision Transformers (ViT) typically require substantial data or strong augmentation/distillation to shine, and can lag CNNs in small-data regimes (Dosovitskiy et al., 2021; Touvron et al., 2021). Consequently, our small-scale experiments should be read primarily as sanity checks rather than ultimate indicators of downstream potential; larger-scale evaluations are a natural next step.

In summary, this report contributes (i) new, training-time formulas to grow the inner dimension k of the attention matrix $QK^\top$ by following functional-gradient signals; and (ii) initial empirical evidence that these formulas behave as intended on controlled benchmarks. By extending growth-based NAS from affine/convolutional layers to attention, we aim to reduce the computational footprint traditionally associated with architecture search while targeting the operator that most differentiates modern sequence and vision models.

2 Modeling

2.1 Notations

We define the following dimensions:

- B the batch size
- e the embedding dimension (also named the "model dimension" in some papers)
- s the sequence length ("token dimension" in some papers)
- k the query/keys dimension
- v the value dimension
- h the number of heads for each multi-head attention block
- w the size of the hidden layer of the MLP in the attention block

Note: As commonly encountered with transformers, we will always have $k < e$ and $v < e$.

We want to work on a vision transformer, in a context of image classification. We will often consider its transformer blocks in isolation from one another. For one particular transformer block, we define the following matrices:

- $X \in \mathbb{R}^{s \times e}$ the input of the transformer block (which represents one image, not a batch). We consider X to be a random variable.
- $i \in [1, ..., h]$ the index of the i-th head.

- For each head, the weight matrices $W_{Q_i}, W_{K_i} \in \mathbb{R}^{e\times k}$ for the "queries/keys", $W_{V_i} \in \mathbb{R}^{e\times v}$ for the "values", $W_{O_i} \in \mathbb{R}^{v\times e}$ for the "output projection". (Note: The matrices $Q = XW_Q, K = XW_K, V = XW_V$ are often defined in papers, we will not do that here for better clarity).
- For each head, a scaling factor κ_i. Note : κ_i is commonly fixed at the value $\sqrt{k}$. As for us, we will change this k dimension, we then either set each κ_i as a learnable parameter, initialized at $\sqrt{k}$ or to a fixed value.
- The weight matrices $W_\xi \in \mathbb{R}^{e\times w}, W_\zeta \in \mathbb{R}^{w\times e}$ for the MLP post-attention block.
- $Y \in \mathbb{R}^{s\times e}$ the output of the transformer block.

2.2 Network

We define the transformer block transformation Block(.) as such, (using the GELU activation function (Hendrycks & Gimpel, 2023)):

$$
\begin{aligned}
L_i(X) &\coloneqq XW_{Q_i}\left(XW_{K_i}\right)^\top, \quad \text{the logits pre-softmax} \\
A_i(X) &= \text{softmax}_{\text{row}}\left(\frac{1}{\kappa_i}L_i(X)\right), \quad \text{the attention matrix} \\
H_i(X) &= A_i(X)XW_{V_i}W_{O_i}, \quad \text{the output of one head} \\
\text{MHA}(X) &= \frac{1}{\sqrt{h}}\sum_i^h H_i(X), \quad \text{the multi-head attention output} \\
\text{Resid}(X) &= X + \text{MHA}(\text{LayerNorm}_1(X)) \\
\text{MLP}(X) &= (\text{GELU}(XW_\xi))W_\zeta \\
\text{Block}(X) &= \text{Resid}(X) + \text{MLP}(\text{LayerNorm}_2(\text{Resid}(X)))
\end{aligned}
$$

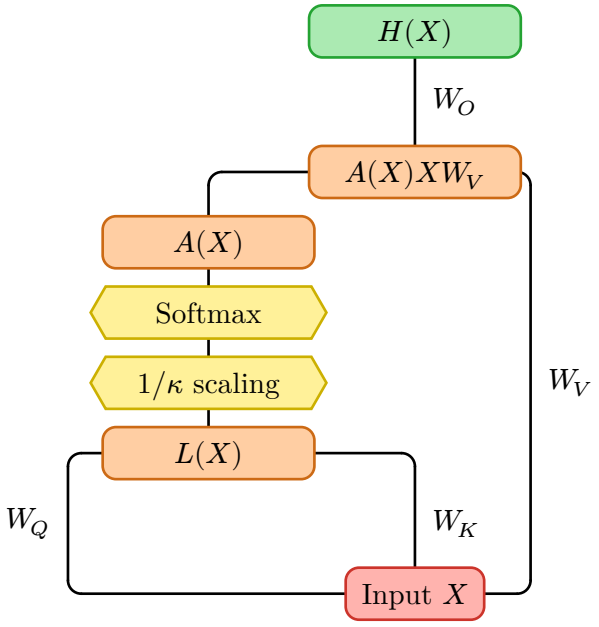

Figure 1: One attention head. Note that the input X already passed through LayerNorm_1, which is not inside the head.

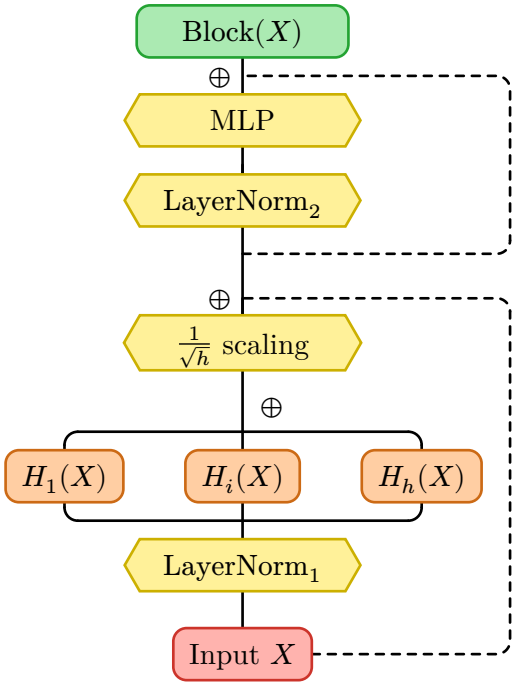

Figure 2: A transformer Block

For the full model, we define:

- $\text{EmbeddingPatcher}(.)$ to transform an image into a sequence of patches (tokens). It is commonly first a convolutional layer with kernel and stride set to (P, P) used to transform the initial image of shape $H \times W \times C$ (height, width, channels) into a sequence of $N = \frac{H}{P} \cdot \frac{W}{P}$ patches, each flattened patch is a vector of length P^2C. Second, we project each patch to the embedding dimension e using a learned linear map.
- $\text{FinalProjection}(.)$ to average-pool the over the tokens then project the result to the classification logits.

The full transformer model (and hence output) would be, for n blocks:

$$\text{FinalProjection}(\text{Block}_n(...\text{Block}_1(\text{EmbeddingPatcher}(\text{image\_input}))))$$

In our case, we only want the embedding patcher to feed the model general "usable" vision data, we do not want the model to learn classification specific to the dataset in the embedding patcher, as we want this to be reserved to the transformer blocks. We will then in practice take a pre-trained embedding patcher, and freeze it during training.

Note: Instead of the average-pooling, it is also possible to append a classification token (CLS) during the embedding, and for the final projection, only use the CLS token.

Note: We will not include a bias for the matrices W_Q, W_K, which can be a valid choice in practice (Padlewski & Djolonga, 2023). To simplify, we haven't included a bias for the other weight matrices in our modeling, but it can easily be added during the implementation.

3 Identifying the optimization problem

In the following, we will only consider one head of a particular transformer block, not the whole model. The input will then be the input after transformation by LayerNorm_1 (Note: the input is the same for each head of a transformer block).

Let

$$L_\theta(X) = XW_QW_K^\top X^\top \in L^2,$$

Parametrized (weights of W_Q, W_K) by a particular $\theta \in \Theta$, with Θ being the set of all possible parametrizations.

Let

$$\mathcal{F}_\theta := \{L_\theta : \theta \in \Theta\}.$$

Furthermore, let $\mathcal{L}(L_\theta) : L^2 \to \mathbb{R}$ be the loss of the full model in function of L_θ, we will consider a functional target T directly linked to the functional gradient, with $\eta > 0$ the "functional training step",

$$T := -\eta \nabla_{L_\theta} \mathcal{L}(L_\theta),$$

we can easily obtain the per-sample evaluation $T(X)$ with backpropagation. We will consider the case $\eta = 1$.

The ideal functional update would be:

$$L^\star = L_\theta + T,$$

but $L^\star \notin \mathcal{F}_\theta$ in most cases.

Let $\mathcal{T}_\theta$ the tangent space of $\mathcal{F}_\theta$ at L_θ, which is the set of all possible infinitesimal variations around L_θ under small parameter variations (Verbockhaven et al., 2024), or the first order approximation of the neighborhood of L_θ in $\mathcal{F}_\theta$.

$$\mathcal{T}_\theta := \left\{ \frac{\partial L_\theta}{\partial \theta} \Delta\theta \mid \Delta\theta \in \Theta \right\}.$$

We want the best output change reachable by the current parameters, i.e., the change in output $\Delta L \in \mathcal{T}_\theta$ closest to T. This leads us to the following L^2 minimization problem:

$$\Delta L^\star = \arg\min_{\Delta L \in \mathcal{T}_\theta} \|T - \Delta L\|_2.$$

Hence, the best change is the orthogonal projection of T onto $\mathcal{T}$. With $R := T - \Delta L^\star$, R and $\Delta L^\star$ are orthogonal.

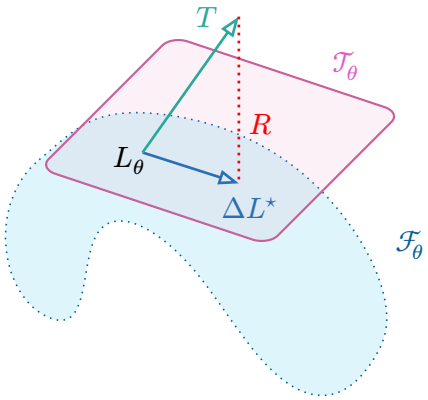

Figure 3: Expressivity bottleneck

4 First method: "Two shots" approach

4.1 Finding the best functional update with a fixed architecture

4.1.1 Objective and first order model

To get the expression of $\Delta L \in \mathcal{T}_\theta$, we can study the change in output of the model following a parameter change $(\Delta W_Q, \Delta W_K)$.

$$\begin{aligned} L^{\text{post-change}}(X) - L(X) &= X(W_Q + \Delta W_Q)(W_K + \Delta W_K)^\top X^\top - X W_Q W_K^\top X^\top \\ &= X(\Delta W_Q W_K^\top + W_Q \Delta W_K^\top + \Delta W_Q \Delta W_K^\top) X^\top. \end{aligned}$$

As we consider the first order change, we drop the quadratic term $\Delta W_Q \Delta W_K$:

$$\Delta L = X(\Delta W_Q W_K^\top + W_Q \Delta W_K^\top) X^\top.$$

We fit $\Delta L(X)$ to $T(X)$ averaged over the dataset $\mathcal{D}$, with n the number of samples, a ridge regularization $\lambda \geq 0$ for numerical stability, and $\|.\|_F$ the Frobenius norm:

$$\Delta W_Q^\star, \Delta W_K^\star = \arg\min_{\Delta W_Q, \Delta W_K} J(\Delta W_Q, \Delta W_K)$$

$$\text{with } J(\Delta W_Q, \Delta W_K) = \frac{1}{2n} \sum_{X \in \mathcal{D}} \|T(X) - \Delta L(X)\|_F^2 + \frac{\lambda}{2} \left(\|\Delta W_Q\|_F^2 + \|\Delta W_K\|_F^2 \right).$$

Note: The projection will not be orthogonal if $\lambda > 0$.

We define the linear operators (for a fixed X):

$$\mathcal{A}_Q(\Delta W_Q) \coloneqq X\Delta W_Q W_K^\top X^\top,\ \mathcal{A}_K(\Delta W_K) \coloneqq X W_Q \Delta W_K^\top X^\top,$$

then

$$\Delta L(X) = \mathcal{A}_Q(\Delta W_Q) + \mathcal{A}_K(\Delta W_K).$$

4.1.2 Adjoints

For linear maps between inner-product spaces $A : \mathcal{U} \to \mathcal{V}$, the adjoint $A^* : \mathcal{V} \to \mathcal{U}$ is the unique map satisfying

$$\langle Au, v\rangle_{\mathcal{V}} = \langle u, A^* v\rangle_{\mathcal{U}} \forall u \in \mathcal{U}, v \in \mathcal{V}.$$

With the Frobenius inner product $\langle A, B\rangle = \operatorname{tr}(A^\top B)$,

$$\begin{aligned}\left\langle \mathcal{A}_Q(\Delta), Y\right\rangle &= \operatorname{tr}\Big(\left(X\Delta W_K^\top X^\top\right)^\top Y\Big)\\ &= \operatorname{tr}(XW_K\Delta^\top X^\top Y) = \operatorname{tr}(\Delta^\top X^\top Y X W_K)\\ &= \langle \Delta, X^\top Y X W_K\rangle.\end{aligned}$$

By the defining property,

$$\mathcal{A}_Q^*(Y) = X^\top Y X W_K \in \mathbb{R}^{e\times k}.$$

Similarly,

$$\begin{aligned}\langle \mathcal{A}_K(\Delta), Y\rangle &= \operatorname{tr}\Big(\left(XW_Q\Delta^\top X^\top\right)^\top Y\Big)\\ &= \operatorname{tr}\Big(X\Delta W_Q^\top X^\top Y\Big) = \operatorname{tr}\big(Y^\top X W_Q \Delta^\top X^\top\big) = \operatorname{tr}\big(\Delta^\top X^\top Y^\top X W_Q\big)\\ &= \big\langle \Delta, X^\top Y^\top X W_Q\big\rangle.\end{aligned}$$

$$\mathcal{A}_K^*(Y) = X^\top Y^\top X W_Q \in \mathbb{R}^{e\times k}.$$

4.1.3 Derivative conventions

We will use the Fréchet differential. For a function f of a matrix variable Z,

- $Df(Z)[H]$ is the linear map in the direction H;
- with the Frobenius inner product $\langle A, B\rangle = \operatorname{tr}(A^\top B)$, the (matrix) gradient $\nabla_Z f$ is defined by

$$Df(Z)[H] = \langle \nabla_Z f, H\rangle \quad \forall H.$$

For multiple variables $Z = (Z_1, Z_2)$ and directions $H = (H_1, H_2)$,

$$Df(Z)[H] \coloneqq D_{Z_1} f[H_1] + D_{Z_2} f[H_2].$$

4.1.4 Gradients

Let $R(X) = \mathcal{A}_Q(\Delta W_Q) + \mathcal{A}_K(\Delta W_K) - T(X)$ the per-sample residual.

We have (Petersen & Pedersen, 2012)

$$\begin{aligned}\mathrm{d}\Big(\frac{1}{2}\|R\|_F^2\Big) = \mathrm{d}\Big(\frac{1}{2}\operatorname{tr}(R^\top R)\Big) &= \frac{1}{2}\operatorname{tr}((\mathrm{d}R)^\top R + R^\top \mathrm{d}R)\\ &= \operatorname{tr}(R^\top \mathrm{d}R) = \langle R, \mathrm{d}R\rangle_F.\end{aligned}$$

We treat J as a function of $\big(\Delta W_Q, \Delta W_K\big)$. For a test direction $\big(H_Q, H_K\big)$,

$$DJ\big[(H_Q, H_K)\big] = \frac{1}{n}\sum_{X\in\mathcal{D}} \big\langle R(X), \mathcal{A}_Q(H_Q) + \mathcal{A}_K(H_K)\big\rangle + \lambda(\big\langle \Delta W_Q, H_Q\big\rangle + \langle \Delta W_K, H_K\rangle).$$

Using the adjoints, we get:

$$DJ\big[(H_Q, H_K)\big] = \left\langle \frac{1}{n}\sum_X \mathcal{A}_Q^*(R(X)) + \lambda \Delta W_Q, H_Q \right\rangle + \left\langle \frac{1}{n}\sum_X \mathcal{A}_K^*(R(X)) + \lambda \Delta W_K, H_K \right\rangle.$$

Therefore the gradients are

$$\nabla_{\Delta W_Q} J = \frac{1}{n}\sum_{X\in\mathcal{D}} \mathcal{A}_Q^*(R(X)) + \lambda \Delta W_Q,\ \nabla_{\Delta W_K} J = \frac{1}{n}\sum_{X\in\mathcal{D}} \mathcal{A}_K^*(R(X)) + \lambda \Delta W_K.$$

4.1.5 Equations

For any linear map $\mathcal{K}$ and $E(Z) = \mathcal{K}(Z) - Y$,

$$\mathrm{d}\Big(\frac{1}{2}\|E\|_F^2\Big) = \langle E, \mathrm{d}E\rangle = \langle E, \mathcal{K}(\mathrm{d}Z)\rangle = \langle \mathcal{K}^*(E), \mathrm{d}Z\rangle \Rightarrow \nabla_Z \frac{1}{2}\|E\|_F^2 = \mathcal{K}^*(E).$$

Let

$$D_{\Delta W_Q} J\big[H_Q\big] \coloneqq DJ\big[(H_Q, 0)\big], \quad D_{\Delta W_K} J[H_K] \coloneqq DJ[(0, H_K)].$$

With $R(X) = \mathcal{A}_Q\big(\Delta W_Q\big) + \mathcal{A}_K(\Delta W_K) - T(X)$ and the adjoints

$$\mathcal{A}_Q^*(Y) = X^\top Y X W_K, \quad \mathcal{A}_K^*(Y) = X^\top Y^\top X W_Q,$$

we have

$$D_{\Delta W_Q} J\big[H_Q\big] = \frac{1}{n}\sum_{X\in\mathcal{D}} \big\langle R(X), \mathcal{A}_Q\big(H_Q\big)\big\rangle + \lambda\big\langle \Delta W_Q, H_Q\big\rangle = \left\langle \frac{1}{n}\sum_X \mathcal{A}_Q^*(R(X)) + \lambda \Delta W_Q, H_Q \right\rangle.$$

Since this must hold for all H_Q, we obtain the first equation

$$\frac{1}{n}\sum_X \mathcal{A}_Q^*(R(X)) + \lambda \Delta W_Q = 0.$$

Expanding $R(X)$ gives

$$\frac{1}{n}\sum_X [\mathcal{A}_Q^*\mathcal{A}_Q\big(\Delta W_Q\big) + \mathcal{A}_Q^*\mathcal{A}_K(\Delta W_K)] + \lambda \Delta W_Q = \frac{1}{n}\sum_X \mathcal{A}_Q^*(T(X)).$$

Similarly, for the keys,

$$D_{\Delta W_K} J[H_K] = \frac{1}{n}\sum_X \langle R(X), \mathcal{A}_K(H_K)\rangle + \lambda\langle \Delta W_K, H_K\rangle = \left\langle \frac{1}{n}\sum_X \mathcal{A}_K^*(R(X)) + \lambda \Delta W_K, H_K \right\rangle,$$

hence for all H_K,

$$\frac{1}{n}\sum_X \mathcal{A}_K^*(R(X)) + \lambda \Delta W_K = 0,$$

i.e.

$$\frac{1}{n}\sum_X [\mathcal{A}_K^* \mathcal{A}_Q(\Delta W_Q) + \mathcal{A}_K^* \mathcal{A}_K(\Delta W_K)] + \lambda \Delta W_K = \frac{1}{n}\sum_X \mathcal{A}_K^*(T(X)).$$

Let $S \coloneqq X^\top X$, we can compute each composed term exactly:

$$\mathcal{A}_Q^* \mathcal{A}_Q(\Delta W_Q) = S\Delta W_Q W_K^\top S W_K, \quad \mathcal{A}_Q^* \mathcal{A}_K(\Delta W_K) = S W_Q \Delta W_K^\top S W_K,$$
$$\mathcal{A}_K^* \mathcal{A}_Q(\Delta W_Q) = S W_K \Delta W_Q^\top S W_Q, \quad \mathcal{A}_K^* \mathcal{A}_K(\Delta W_K) = S\Delta W_K W_Q^\top S W_Q,$$
$$\mathcal{A}_Q^*(T(X)) = X^\top T(X) X W_K, \quad \mathcal{A}_K^*(T(X)) = X^\top T(X)^\top X W_Q.$$

Therefore we get the equations, with $\bar{C} \coloneqq \frac{1}{n}\sum_X X^\top T(X) X$

$$\frac{1}{n}\sum_X (S\Delta W_Q W_K^\top S W_K) + \frac{1}{n}\sum_X (S W_Q \Delta W_K^\top S W_K) + \lambda \Delta W_Q = \bar{C} W_K, \tag{1}$$

$$\frac{1}{n}\sum_X \left(S W_K \Delta W_Q^\top S W_Q\right) + \frac{1}{n}\sum_X \left(S\Delta W_K W_Q^\top S W_Q\right) + \lambda \Delta W_K = \bar{C}^\top W_Q. \tag{2}$$

4.1.6 Vectorized linear system

We will use vec(.) to reshape a matrix into a vector by stacking its columns. For a matrix $A \in \mathbb{R}^{m\times n}$, $\mathrm{vec}(A) = \begin{pmatrix} A_{:,1} \\ \vdots \\ A_{:,n} \end{pmatrix}$.

For each sample X we define

$$B_Q \coloneqq S W_Q \in \mathbb{R}^{e\times k}, \quad B_K \coloneqq S W_K \in \mathbb{R}^{e\times k},$$
$$G_Q(X) \coloneqq W_Q^\top S W_Q = G_Q(X)^\top \in \mathbb{R}^{k\times k}, \quad G_K(X) \coloneqq W_K^\top S W_K = G_K(X)^\top \in \mathbb{R}^{k\times k},$$
$$w_Q \coloneqq \mathrm{vec}\left(\Delta W_Q\right), \quad w_K \coloneqq \mathrm{vec}(\Delta W_K) \in \mathbb{R}^{ek} \quad w = \begin{pmatrix} w_Q \\ w_K \end{pmatrix},$$
$$b_Q \coloneqq \mathrm{vec}\left(\bar{C} W_K\right), \quad b_K \coloneqq \mathrm{vec}\left(\bar{C}^\top W_Q\right), \quad b = \begin{pmatrix} b_Q \\ b_K \end{pmatrix}.$$

Let $\otimes$ be the Kronecker product, we have the identities (Petersen & Pedersen, 2012):

$$\mathrm{vec}(U\Delta V) = (V^\top \otimes U)\ \mathrm{vec}(\Delta), \quad \mathrm{vec}(U\Delta^\top V) = (V^\top \otimes U) K_{e,k}\ \mathrm{vec}(\Delta).$$

We define the commutation matrix $K_{m,n} \in \mathbb{R}^{mn\times mn}$ such that for a matrix $M \in \mathbb{R}^{m\times n}$, $K_{m,n}\ \mathrm{vec}(M) = \mathrm{vec}(M^\top)$ (Magnus & Neudecker, 1979).

Hence for matrices A, B, C with $B \in \mathbb{R}^{e\times k}$, we have the identity:

$$\begin{aligned}\mathrm{vec}(AB^\top C) &= (C^\top \otimes A)\mathrm{vec}(B^\top) \\ &= (C^\top \otimes A) K_{e,k}\ \mathrm{vec}(B).\end{aligned}$$

(1) is equivalent to:

$$\frac{1}{n}\sum_X \mathrm{vec}\left(S\Delta W_Q G_K\right) + \frac{1}{n}\sum_X \mathrm{vec}\left(B_Q \Delta W_K^\top B_K\right) + \lambda w_Q = b_Q$$
$$\iff \frac{1}{n}\sum_X (G_K \otimes S) w_Q + \frac{1}{n}\sum_X (B_K^\top \otimes B_Q) K_{e,k} w_K + \lambda w_Q = b_Q$$

(2) is equivalent to:

$$\frac{1}{n}\sum_X \mathrm{vec}\big(B_K \Delta W_Q^\top B_Q\big) + \frac{1}{n}\sum_X \mathrm{vec}\big(S\Delta W_K G_Q\big) + \lambda w_K = b_K$$
$$\iff \frac{1}{n}\sum_X \big(B_Q^\top \otimes B_K\big) K_{e,k} w_Q + \frac{1}{n}\sum_X (G_Q \otimes S) w_K + \lambda w_K = b_K$$

Hence we have

$$\Omega := \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \in \mathbb{R}^{(2ek)\times(2ek)}$$
$$\Omega w = b$$

with

$$A_{11} := \lambda I_{ek} + \frac{1}{n}\sum_X (G_K \otimes S), \quad A_{12} := \frac{1}{n}\sum_X (B_K^\top \otimes B_Q) K_{e,k},$$
$$A_{21} := \frac{1}{n}\sum_X \big(B_Q^\top \otimes B_K\big) K_{e,k}, \quad A_{22} := \lambda I_{e,k} + \frac{1}{n}\sum_X (G_Q \otimes S).$$

4.1.7 Symmetry of Ω

For arbitrary matrices A, B we have the identity $(A \otimes B)^\top = A^\top \otimes B^\top$ (Petersen & Pedersen, 2012), hence A_{11} and A_{22} are symmetric.

For arbitrary matrices $A, B \in \mathbb{R}^{e\times k}$ we have (Magnus & Neudecker, 1979):

$$K_{k,e}(A \otimes B) = (B \otimes A) K_{e,k} \quad \text{and} \quad \big(K_{e,k}\big)^\top = K_{k,e}.$$

We then have

$$\big((B_K^\top \otimes B_Q) K_{e,k}\big)^\top = K_{k,e}\big(B_K \otimes B_Q^\top\big)$$
$$= \big(B_Q^\top \otimes B_K\big) K_{e,k}$$

Which implies

$$A_{12}^\top = A_{21}$$

Hence Ω is symmetric.

4.1.8 Positive definiteness of Ω

Let

$$\mathcal{U} := \mathbb{R}^{e\times k} \times \mathbb{R}^{e\times k}, \quad \mathcal{V} := \mathbb{R}^{s\times s}.$$

We endow $\mathcal{U}$ with the product Frobenius inner product

$$\langle (A_1, A_2), (B_1, B_2) \rangle_{\mathcal{U}} := \langle A_1, B_1 \rangle + \langle A_2, B_2 \rangle,$$

so that $\|(A_1, A_2)\|_{\mathcal{U}}^2 = \|A_1\|_F^2 + \|A_2\|_F^2$. The space $\mathcal{V}$ uses the Frobenius inner product $\langle Y, Z \rangle_{\mathcal{V}} := \langle Y, Z \rangle$.

For each sample X, we define the linear map $\mathcal{M}_X : \mathcal{U} \to \mathcal{V}$ by

$$\mathcal{M}_X(\Delta W_Q, \Delta W_K) := \mathcal{A}_Q(\Delta W_Q) + \mathcal{A}_K(\Delta W_K) = X \Delta W_Q W_K^\top X^\top + X W_Q \Delta W_K^\top X^\top.$$

Since the spaces are finite-dimensional inner-product spaces, the adjoint $\mathcal{M}_X^* : \mathcal{V} \to \mathcal{U}$ exists and is uniquely defined by

$$\langle \mathcal{M}_X U, V \rangle_{\mathcal{V}} = \langle U, \mathcal{M}_X^* V \rangle_{\mathcal{U}}.$$

Using the adjoints already computed,

$$\mathcal{A}_Q^*(Y) = X^\top Y X W_K, \quad \mathcal{A}_K^*(Y) = X^\top Y^\top X W_Q,$$

we have

$$\mathcal{M}_X^*(Y) = \big(\mathcal{A}_Q^*(Y),; \mathcal{A}_K^*(Y)\big).$$

We write the objective in the variable $\Delta := \big(\Delta W_Q, \Delta W_K\big) \in \mathcal{U}$:

$$J(\Delta) = \frac{1}{2n} \sum_X \|\mathcal{M}_X \Delta - T(X)\|_F^2 + \frac{\lambda}{2} \|\Delta\|_{\mathcal{U}}^2. \tag{3}$$

We can expand each squared norm and use the adjoint identity

$$\|\mathcal{M}_X \Delta\|_F^2 = \langle \mathcal{M}_X \Delta, \mathcal{M}_X \Delta \rangle_F = \langle \Delta, \mathcal{M}_X^* \mathcal{M}_X \Delta \rangle_{\mathcal{U}}$$

then

$$\begin{aligned} \|\mathcal{M}_X \Delta - T(X)\|_F^2 &= \|\mathcal{M}_X \Delta\|_F^2 - 2\langle T(X), \mathcal{M}_X \Delta \rangle_F + \|T(X)\|_F^2 \\ &= \langle \Delta, \mathcal{M}_X^* \mathcal{M}_X \Delta \rangle_{\mathcal{U}} - 2\langle T(X), \mathcal{M}_X \Delta \rangle_F + \|T(X)\|_F^2 \end{aligned}$$

to obtain

$$\begin{aligned} J(\Delta) &= \frac{1}{2} \left\langle \Delta, \left(\lambda I_{\mathcal{U}} + \frac{1}{n} \sum_X \mathcal{M}_X^* \mathcal{M}_X \right) \Delta \right\rangle_{\mathcal{U}} - \left\langle \frac{1}{n} \sum_X \mathcal{M}_X^* T(X), \Delta \right\rangle_{\mathcal{U}} + \text{const}(T). \\ &= \frac{1}{2} \langle \Delta, \Omega \Delta \rangle_{\mathcal{U}} - \langle b, \Delta \rangle_{\mathcal{U}} + \text{const} \end{aligned}$$

Hence $\nabla J = \Omega \Delta$ and the (constant) Hessian is

$$\nabla^2 J = \Omega := \lambda I_{\mathcal{U}} + H, \quad \text{with} \quad H := \frac{1}{n} \sum_X \mathcal{M}_X^* \mathcal{M}_X.$$

For any $U, V \in \mathcal{U}$, by definition,

$$\langle U, (\mathcal{M}_X^* \mathcal{M}_X) V \rangle_{\mathcal{U}} = \langle \mathcal{M}_X U, \mathcal{M}_X V \rangle_F = \langle (\mathcal{M}_X^* \mathcal{M}_X) U, V \rangle_{\mathcal{U}},$$

so $\mathcal{M}_X^* \mathcal{M}_X$ is self-adjoint. Moreover,

$$\langle U, (\mathcal{M}_X^* \mathcal{M}_X) U \rangle_{\mathcal{U}} = \|\mathcal{M}_X U\|_F^2 \geq 0,$$

so $\mathcal{M}_X^* \mathcal{M}_X \succeq 0$. Averaging preserves these properties; hence $H \succeq 0$ and H is self-adjoint. Adding $\lambda I_{\mathcal{U}}$ preserves symmetry, so Ω is symmetric as already proven.

Moreover for any nonzero $\Delta \in \mathcal{U}$ and $\lambda > 0$,

$$\langle \Delta, \Omega \Delta \rangle_{\mathcal{U}} = \lambda \|\Delta\|_{\mathcal{U}}^2 + \frac{1}{n} \sum_X \|\mathcal{M}_X \Delta\|_F^2 \geq \lambda \|\Delta\|_{\mathcal{U}}^2 > 0,$$

Therefore $\Omega \succ 0$, the matrix is positive definite if $\lambda > 0$.

Moreover, for $\lambda > 0$, with (3), we can see that

$$\lim_{\|\Delta\| \to \infty} J(\Delta) = \infty,$$

hence J is coercive, and admits a minimizer.

From $\nabla^2 J \succ 0$, we can conclude J is strictly convex, hence the minimizer is unique.

4.1.9 Solving with a Cholesky decomposition

As Ω is symmetric positive definite, we can apply a Cholesky decomposition to Ω:

$$\Omega = LL^\top,$$

L is lower triangular and its diagonal values are strictly positive,

$$L = \begin{pmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{pmatrix}, \quad L^\top = \begin{pmatrix} L_{11}^\top & L_{21}^\top \\ 0 & L_{22}^\top \end{pmatrix}, \quad L_{11}, L_{22}, L_{21} \in \mathbb{R}^{(ek)\times(ek)}.$$

Let $y = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}$, we first solve

$$Ly = b \iff \begin{pmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} b_Q \\ b_K \end{pmatrix} \Rightarrow \begin{cases} L_{11} y_1 = b_Q \\ L_{21} y_1 + L_{22} y_2 = b_K \end{cases} \Rightarrow \begin{cases} y_1 = L_{11}^{-1} b_Q \\ y_2 = L_{22}^{-1}(b_K - L_{21} y_1) \end{cases}$$

then,

$$L^\top w = y \iff \begin{pmatrix} L_{11}^\top & L_{21}^\top \\ 0 & L_{22}^\top \end{pmatrix} \begin{pmatrix} w_Q \\ w_K \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} \Rightarrow \begin{cases} L_{22}^\top w_K = y_2 \\ L_{11}^\top w_Q + L_{21}^\top w_K = y_1 \end{cases} \Rightarrow \begin{cases} w_K = (L_{22}^\top)^{-1} y_2 \\ w_Q = (L_{11}^\top)^{-1} \Big(y_1 - L_{21}^\top (L_{22}^\top)^{-1} y_2 \Big) \end{cases}$$

For any vector $v \in \mathbb{R}^{mn}$, we define

$$\mathrm{unvec}_{m\times n}(v) = A \in \mathbb{R}^{m\times n} \text{ such that } \mathrm{vec}(A) = v.$$

We finally get

$$\Delta W_Q^\star = \mathrm{unvec}_{e\times k}\big(w_Q\big), \quad \Delta W_K^\star = \mathrm{unvec}_{e\times k}(w_K),$$

and the best functional change in output with the current architecture:

$$\Delta L^\star(X) = X\Big(\Delta W_Q^\star W_K^\top + W_Q \Delta W_K^\top\Big) X^\top.$$

Note: It could be possible to get a faster and less memory intensive solution by doing a block Cholesky via Schur complement instead.

4.1.10 Notes on implementation

4.1.10.1 Choice of λ to be scale-aware

Let

$$\bar{\sigma} := \frac{1}{2}\left(\frac{\mathrm{tr}(A_{11})}{ek} + \frac{\mathrm{tr}(A_{22})}{ek} \right),$$

we will choose λ such that

$$\lambda = \tau \bar{\sigma}, \quad \tau \in [10^{-3}, 10^{-1}].$$

4.1.10.2 Avoid forming $K_{e,k}$

Let M, N be matrices, $K_{e,k}$ a commutation matrix such that $M = NK_{e,k}$. We have

$$M = N[:, P]$$

where P is a list of permutation indices that implements right-multiplication by $K_{e,k}$ and $N[:, P]$ is the matrix N with permuted columns according to the index list P.

See Appendix A.1 for the code snippets to get P, to do the usual vec(.) and unvec(.).

4.1.10.3 Algorithm

Algorithm 1: Get $\Delta W_Q^\star, \Delta W_K^\star$

Input: Dataset $\mathcal{D}$ (input X, target Y, n samples), Model $\mathcal{M}$

Initialize all $(.)^{\text{sum}} \leftarrow 0$

for X, Y in $\mathcal{D}$

  $\mathcal{M}$ forward to access S, B_Q, B_K, G_Q, G_K

  $\mathcal{M}$ backward to access T

  $A_{11}^{\text{sum}} \leftarrow A_{11}^{\text{sum}} + G_K \otimes S, \quad A_{22}^{\text{sum}} \leftarrow A_{22}^{\text{sum}} + G_Q \otimes S$

  $A_{12}^{\text{sum}} \leftarrow A_{12}^{\text{sum}} + B_K^\top \otimes B_Q, \quad C^{\text{sum}} \leftarrow C^{\text{sum}} + X^\top T X$

Get all statistics $\bar{(.)} \leftarrow \frac{(.)^{\text{sum}}}{n}$

$P \leftarrow$ K_perm_indices()

$A_{12} \leftarrow A_{12}[:, P]$

$A_{\text{ij}} \leftarrow \frac{1}{2}\left(A_{\text{ij}} + A_{\text{ij}}^\top\right)$ for ij $= [11, 22]$ (avoid precision errors)

$\bar{A}_{11} \leftarrow \bar{A}_{11} + \lambda I, \quad \bar{A}_{22} \leftarrow \bar{A}_{22} + \lambda I$

$\Omega \leftarrow \begin{pmatrix} \bar{A}_{11} & \bar{A}_{12} \\ \bar{A}_{12}^\top & \bar{A}_{22} \end{pmatrix}, \quad b = \begin{pmatrix} \text{vec}\left(\bar{C} W_K\right) \\ \text{vec}\left(\bar{C}^\top W_Q\right) \end{pmatrix}$

$L \leftarrow$ Cholesky_decomposition(Ω)

$\begin{pmatrix} w_Q \\ w_K \end{pmatrix} \leftarrow$ Cholesky_solve(b, L)

$\Delta W_Q^\star \leftarrow \text{unvec}_{e \times k}\left(w_Q\right), \quad \Delta W_K^\star \leftarrow \text{unvec}_{e \times k}(w_K)$

Note: The dataset chosen can be either the train or the validation dataset, and we can choose to iterate over a fraction of it instead of the whole.

4.2 Growing the k dimension

4.2.1 Modeling

To make the k dimension grow, we want to find p new columns to concatenate into W_Q and W_K:

$$W_Q^{\text{post-growth}} = \left[W_Q + \Delta W_Q^\star \mid \widetilde{W}_Q\right]$$
$$W_K^{\text{post-growth}} = \left[W_K + \Delta W_K^\star \mid \widetilde{W}_K\right]$$

with, for the queries (same for keys), $W_Q^{\text{post-growth}} \in \mathbb{R}^{e \times (k+p)}$, and the new matrix $\widetilde{W}_Q \in \mathbb{R}^{e \times p}$.

Let

$$Z_0 := \Delta W_Q^\star W_K^\top + W_Q \Delta W_K^{\star\top} \in \mathbb{R}^{e \times e},$$

then the per-sample residual is $T(X) - X Z_0 X^\top$, with $X Z_0 X^\top$ the “already realized” first-order contribution.

We seek a growth matrix $Z = \widetilde{W}_Q \widetilde{W}_K^\top$ such that $X(Z_0 + Z)X^\top$ matches the functional target $T(X)$.

We introduce the centered variable

$$Y \coloneqq Z_0 + Z \in \mathbb{R}^{e \times e}.$$

We solve the centered ridge problem

$$Y^\star = \arg\min_{Y \in \mathbb{R}^{e \times e}} F(Y),$$

$$\text{with } F(Y) \coloneqq \frac{1}{2n} \sum_X \left\| T(X) - XYX^\top \right\|_F^2 + \frac{\alpha}{2} \|Y - Z_0\|_F^2, \quad \alpha > 0.$$

The actual objective being $Z^\star = Y^\star - Z_0$.

4.2.2 Solving

Recall $S = X^\top X$, $C = \frac{1}{n} \sum_X X^\top T(X) X$, and let

$$\mathcal{H}(Y) \coloneqq \frac{1}{n} \sum_X SYS.$$

The gradient is

$$\nabla F(Y) = \mathcal{H}(Y) - C + \alpha(Y - Z_0).$$

The first-order condition $\nabla F(Y) = 0$ gives the equation

$$\mathcal{H}(Y^\star) + \alpha Y^\star = C + \alpha Z_0.$$

We can vectorize using $\operatorname{vec}(UYV) = (V^\top \otimes U) \operatorname{vec}(Y)$ to solve, we define

$$H_0 \coloneqq \frac{1}{n} \sum_X (S \otimes S), \quad H_\alpha \coloneqq H_0 + \alpha I.$$

Since each $S = S^\top$ and $S \succeq 0$, H_0 is symmetric positive semi-definite, and H_α is symmetric positive definite for $\alpha > 0$.

The linear system to solve is

$$H_\alpha \operatorname{vec}(Y^\star) = \operatorname{vec}(C) + \alpha \operatorname{vec}(Z_0), \quad Z^\star = Y^\star - Z_0,$$

Which we can solve with a Cholesky decomposition.

Note: The statistic H_0 can be computed while computing the statistics of Section 4, where C is already computed, so the additional cost is low.

4.2.3 Creating the new matrices

The best rank-p approximation of $Z^\star$ is given by its truncated singular value decomposition (SVD, we select the first p columns) (Eckart & Young, 1936; Mirsky, 1960):

$$Z^\star = U\Sigma V^\top, \quad \widetilde{W}_Q = U\Sigma^{\frac{1}{2}}[:, 1:p], \quad \widetilde{W}_K = V\Sigma^{\frac{1}{2}}[:, 1:p].$$

For the choice of p, let $\sigma_1 \geq \sigma_2 \geq \ldots$ be the ordered singular values of $Z^\star$ from the SVD. We choose the smallest p such that:

$$\frac{\sum_{j\leq p}\sigma_j^2}{\sum_j \sigma_j^2} \geq \beta, \quad \beta \in [0.95, 0.99].$$

We initialize $\alpha = \tau' \frac{1}{n} \sum_X \left\| X^\top X \right\|_F^2, \quad \tau' \in [10^{-3}, 10^{-1}]$.

Note: To solve the problem in this chapter, we chose to solve with a non iterative algorithm, yielding a closed form. This however doesn't seem too scalable, as it introduces huge matrices for each head ($H_0 \in \mathbb{R}^{e^2 \times e^2}$) and quickly cause memory issues for large values of e. In practice, we would prefer an iterative algorithm to not have to store this matrix. This remark can also be valid for the problem of Section 4, albeit less so, as the kronecker matrices of Section 4 are of size $ek \times ek$, with $k \ll e$.

4.3 Growth criterion

A simple growth criterion to consider could be, for each head, its mean residual between the functional target and the change in output of the best update at a fixed architecture.

$$\bar{R} = \frac{1}{n} \sum_X \left\| T(X) - \Delta L^\star(X) \right\|_F.$$

A high value corresponds to a head that lacks capacity to reach T, i.e. has an important “expressivity bottleneck” (Verbockhaven et al., 2024).

We can choose this $\bar{R}$ as a criterion to choose where to grow, selecting for example the head with the largest value. It also could answer the question of “when”, as it is possible to fix a threshold, evaluate $\bar{R}$ for each head every k training step, and growing any head going over the threshold.

However, it has some limits:

- It considers the residual in function of the change in output at the first order, which is not exactly the actual change realized by the model.
- It doesn't directly consider the loss $\mathcal{L}$, which is the actual “final” objective. It incorporates some information about it via $T(X)$, as large values of $\|T(X)\|$ hint at a higher loss reduction,(with a fixed angle between $T(X)$ and $\Delta L^\star$, higher $\|T(X)\|$ leads to higher $\|T(X) - \Delta L^\star(X)\|$).
- It doesn't have information about the quality of the growth, i.e. if the growth doesn't help to reduce the expressivity bottleneck, the algorithm will keep selecting the head for growth, without effect on the loss.

To consider a more complex criterion, we define the residual mean post-growth, and the mean norm of the functional target,

$$\bar{r} := \frac{1}{n} \sum_X \left\| T(X) - X(Z_0 + Z^\star)X^\top \right\|_F, \quad \bar{T} := \frac{1}{n} \sum_X \left\| T(X) \right\|_F.$$

A better criterion could be the ratio of the residuals pre/post growth, scaled by the norm of the functional target.

$$\frac{\bar{R}}{\bar{r}} \cdot \bar{T}$$

This criterion has the information of how much the growth helped close the expressivity bottleneck, scaled by the mean of the functional gradient norm to give more importance to heads with a higher potential for loss reduction.

4.4 Training algorithm

A training loop with growth could then follow this example algorithm, which gives answers to when, where and how to grow.

Algorithm 2: Training loop

Input: Dataset $\mathcal{D}$ (input X, target Y, n samples), Model $\mathcal{M}$

```
for X, Y in D
    REGULAR_TRAINING()
    After the last batch of each epoch, do:
        ACCUMULATE_STATISTICS()
        For each head:
            COMPUTE ΔW̃*_Q, ΔW̃*_K()
            COMPUTE W̃_Q, W̃_K()
        COMPUTE_RESIDUALS()
        SELECT_HEAD_TO_GROW_AND_APPLY()
```

5 Second method: “One shot” approach

5.1 Solving the problem

There is another way to approach the problem. Instead of finding the best update at fixed architecture, then growing, we consider the initial weight matrices as a product $P := W_Q W_K^\top$, and we search for the best update $\Delta P^\star$ to fit $T(X)$. This skips the step of finding the best update at fixed architecture.

$$\Delta P^\star := \arg\min_{\Delta P \in \mathbb{R}^{e\times e}} \frac{1}{2n}\sum_X \left\|T(X) - X\Delta P X^\top\right\|_F^2 + \frac{\alpha}{2}\|\Delta P\|_F^2, \quad \alpha > 0,$$

which leads to

$$H_\alpha \operatorname{vec}(\Delta P^\star) = \operatorname{vec}(C).$$

Hence this method has the advantage of just needing two statistics, C and H_0.

However we cannot split $\Delta P^\star$ with a truncated SVD; if we were to do that, we would get:

$$\Delta P^\star = U\Sigma V^\top, \quad \widetilde{W}_Q = U\Sigma^{\frac{1}{2}}[:, 1:k+p], \quad \widetilde{W}_K = V\Sigma^{\frac{1}{2}}[:, 1:k+p].$$

$$W_Q \in \mathbb{R}^{e\times k}, \ \widetilde{W}_Q \in \mathbb{R}^{e\times(k+p)}, \ \text{ incompatible.}$$

We have to act on $P + \Delta P^\star$ (truncated SVD, we select the first $k+p$ columns):

$$(P + \Delta P^\star) = U\Sigma V^\top, \quad W_Q^{\text{post-growth}} = U\Sigma^{\frac{1}{2}}[:, 1:k+p], \quad W_K^{\text{post-growth}} = V\Sigma^{\frac{1}{2}}[:, 1:k+p].$$

We directly get new “end-result” matrices without an incremental update. As the factorization is not unique, the weights composing the new weight matrices can radically change, compared to the first method which had a weight update + concatenation of new weights. As a Plain SGD/ Adam aren’t reparameterization-invariant (Martens, 2020; Neyshabur et al., 2015), different factorizations lead to different update geometry. Hence with this method, we alter the model more than with the first method, which could lead to instability.

Furthermore, we do not have access to $\bar{R}$ to compute the growth criterion. We then propose another one, let

$$P^{\text{post-growth}} = \text{SVD}_{\text{trunc } k+p}(P + \Delta P^\star) = W_Q^{\text{post-growth}} W_K^{(\text{post-growth})^\top},$$
$$\Delta L_p(X) = X\big(P^{\text{post-growth}} - P\big)X^\top,$$

$\Delta L_p(X)$ being the change in output per-sample between after and before the growth (caution, $\Delta L_p(X) \neq X\Delta P^\star X^\top$ if $(k+p) < \text{rank}(P + \Delta P^\star)$).

Let

$$\bar{F} \coloneqq \frac{1}{n}\sum_X \big\langle T(X), \Delta L_p(X)\big\rangle, \quad \bar{L} \coloneqq \frac{1}{n}\sum_X \big\|\Delta L_p(X)\big\|_F, \quad \bar{T} = \frac{1}{n}\sum_X \|T(X)\|_F.$$

and with $\text{cs}(.,.)$ the cosine similarity, $\bar{E} = \frac{1}{n}\sum_X \text{cs}\big(T(X), \Delta L_p(X)\big)$.

The criterion would be

$$\frac{\bar{F}}{\bar{L}} = \bar{E} \cdot \bar{T},$$

representing the angle between the functional target and the change in output, scaled by the mean norm of the functional target to give the information of potential loss reduction.

Since this method is not able compute $\bar{R}$, we do not have information on the expressivity bottleneck during the pre-growth state, this could lead to problems where a head with a low $\bar{R}$ pre-growth keeps getting selected. Updating this head would give a good loss reduction, but the growth wouldn't really bring anything useful, and it could “steal” the spot of a better growth candidate.

5.2 Loss reduction

As of now, we have never scaled the new computed weights before integrating them to the model. Since we already have the update computed, we could search for a λ solving:

$$\arg\min_{\lambda \in \mathbb{R}} \mathcal{L}\big(L + \lambda \Delta L_p\big).$$

Let the functional inner product be $\langle A, B\rangle_{L^2} \coloneqq \frac{1}{n}\sum_X \text{tr}\big(A(X)^\top B(X)\big)$, and

$$\varphi(\lambda) \coloneqq \mathcal{L}\big(L + \lambda \Delta L_p\big)$$

According to the Taylor expansion around $\lambda_0 = 0$,

$$\begin{aligned}
\varphi'(0) &= \big\langle \nabla_L \mathcal{L}(L), \Delta L_p\big\rangle_{L^2} \\
&= -\big\langle T, \Delta L_p\big\rangle_{L^2} \\
&= -\frac{1}{n}\sum_X \big\langle T(X), X\big(P^{\text{post-growth}} - P\big)X^\top\big\rangle_F \\
&= -\frac{1}{n}\sum_X \big\langle X^\top T(X) X, \big(P^{\text{post-growth}} - P\big)\big\rangle_F \\
&= -\big\langle C, \big(P^{\text{post-growth}} - P\big)\big\rangle_F
\end{aligned}$$

Using $\varphi'(0)$, we can apply an iterative backtracking line-search (Boyd & Vandenberghe, 2004) to find a $\hat{\lambda}$ estimating $\arg\min_\lambda \varphi(\lambda)$. See Figure 4 for an example.

Algorithm 3: Backtracking line-search

Input: Function φ, starting point λ_0, first order improvement $\varphi'(0)$,parameters $\alpha \in (0,1)$, $\beta \in (0,1)$

If $\varphi'(0) \geq 0$:
  return 0
$\lambda \leftarrow \lambda_0$
for $t = 1, ..., \text{max\_iter}$:
  if $\varphi(\lambda) \leq \varphi(0) + \lambda \cdot \alpha \cdot \varphi'(0)$:
    return λ
  else: $\lambda \leftarrow \beta \cdot \lambda$
return λ

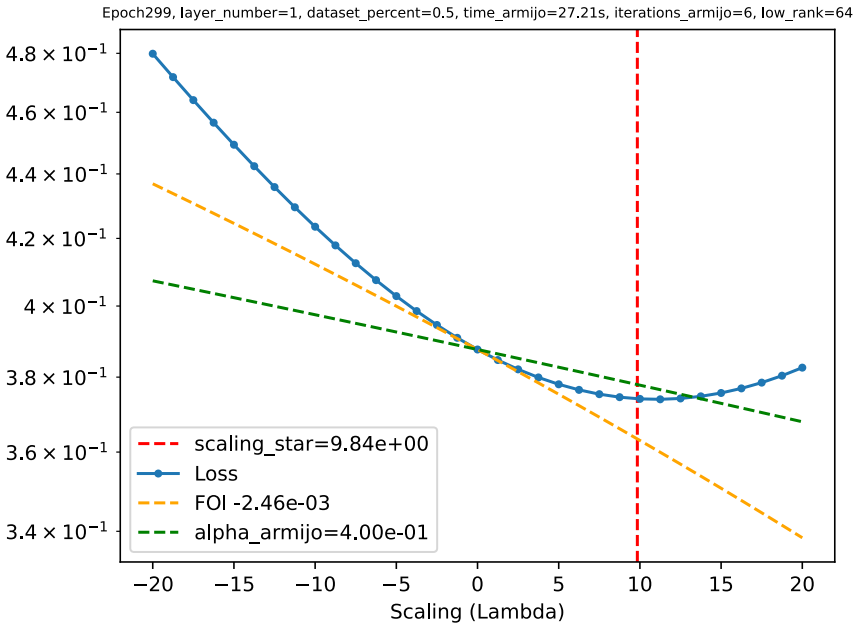

Figure 4: **Plot of $\varphi(\lambda)$ (blue line) with a backtracking line-search.** Yellow line: Tangent at 0, coefficient $\varphi'(0)$. Green line: Same as yellow with its coefficient scaled by α, acts as a threshold. Red value: $\hat{\lambda}$ returned.

6 Experimentations

6.1 Convergence of the statistics

We studied the empirical convergence of the two statistics H_0 and C used for the “One-shot” approach, on the CIFAR10 dataset (Krizhevsky et al., 2009). We iterated over the 390 batches (50k samples) with a batch size of 128. The x axis is the number of cumulative batches used to estimate the tested statistic (except for the “norm” graph). The target statistic will always be the statistic estimated over 390 batches. Let $\alpha(x)$ be the computed statistic, and β the target statistic.

Figure 5 represents the convergence of C, Figure 6 represents the convergence of H_0.

We studied multiple quantities:

- Norm: $f(x) = \|\alpha(x)\|_F$
- Norm ratio: $f(x) = \frac{\|\alpha(x)\|_F}{\|\beta\|_F}$
- Cosine similarity: $f(x) = \frac{\langle \alpha(x), \beta \rangle_F}{\|\alpha(x)\|_F \cdot \|\beta\|_F}$
- RMSE: $f(x) = \|\alpha(x) - \beta\|_F^2$
- Relative RMSE: $f(x) = \frac{\|\alpha(x) - \beta\|_F^2}{\|\beta\|_F^2}$

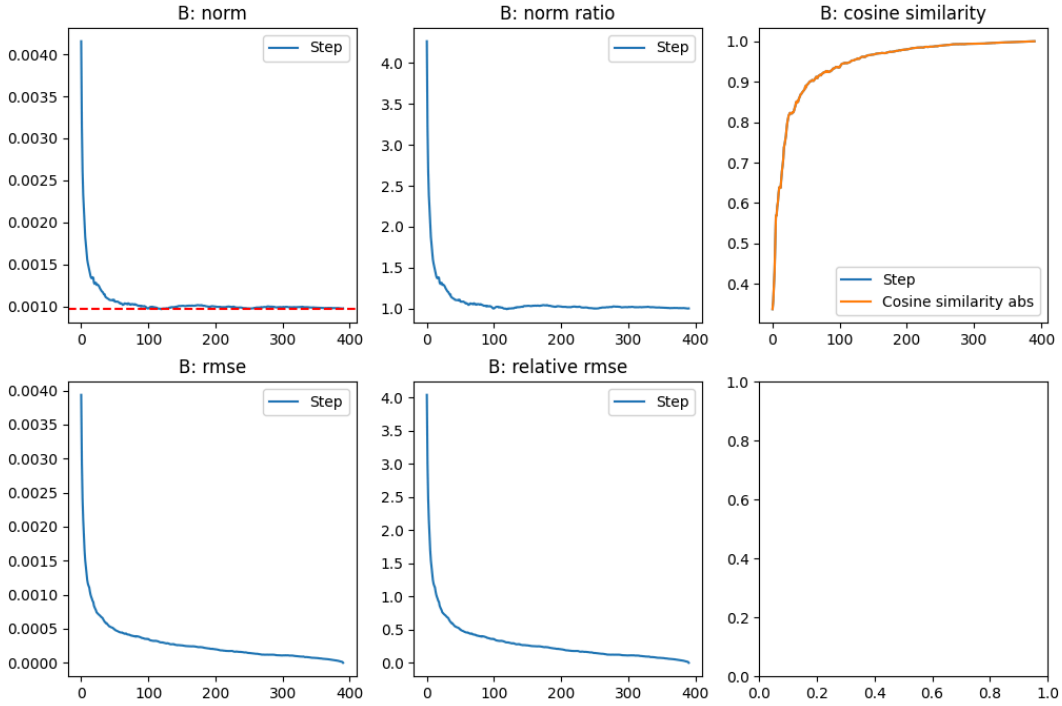

Figure 5: Convergence of C

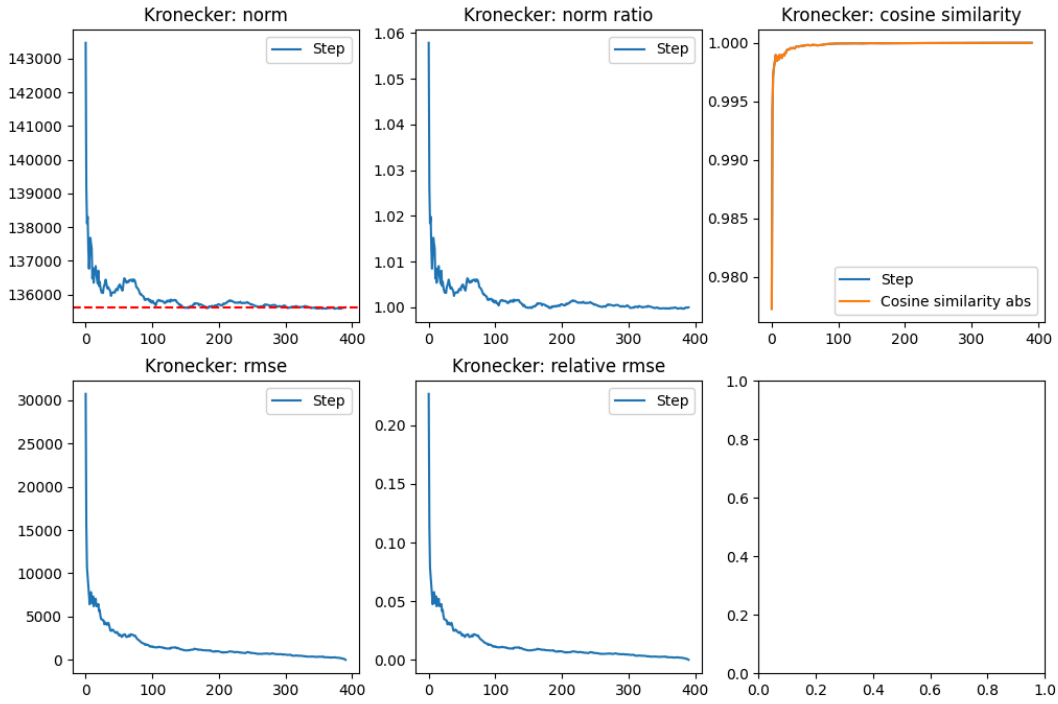

Figure 6: Convergence of H_0

We can see on both figures that, although the convergence of C seem a bit slower compared to H_0, both seem to converge quite fast.

As iterating over the whole dataset is costly, we can consider using a fraction of it to compute the statistics. Moreover, we can also compute the statistics over the validation set instead of the training set, to avoid overfitting the growth to the training set.

6.2 Significativity of k in the loss

Unfortunately, we were not able to see a significant impact of the k dimension in the loss reduction. We experimented on CIFAR100 (Krizhevsky et al., n.d.) and Imagenette (Howard, 2019), a subset of ImageNet with only 10 classes.

Even when restricting other learnable parameters (freezing the embedding patcher pre-trained on another dataset, restricting the size of the hidden layer of $W_V W_O, W_\xi W_\zeta$ to 1), we were not able to see a significant impact of k in the loss reduction. We hypothesize that the datasets and models used were not complex enough to have a need of an important k, an issue that could be linked to the general underperformance of vanilla ViT in small scale (Dosovitskiy et al., 2021).

We can see on Figure 7 that the test accuracy on Imagenette, without growth, is quite similar for $k = 1$ and $k = 16$, even with the MLP hidden sizes set to 1. In our experiments, the difference between between the two curves decreases as we allow the MLPs to have more parameters.

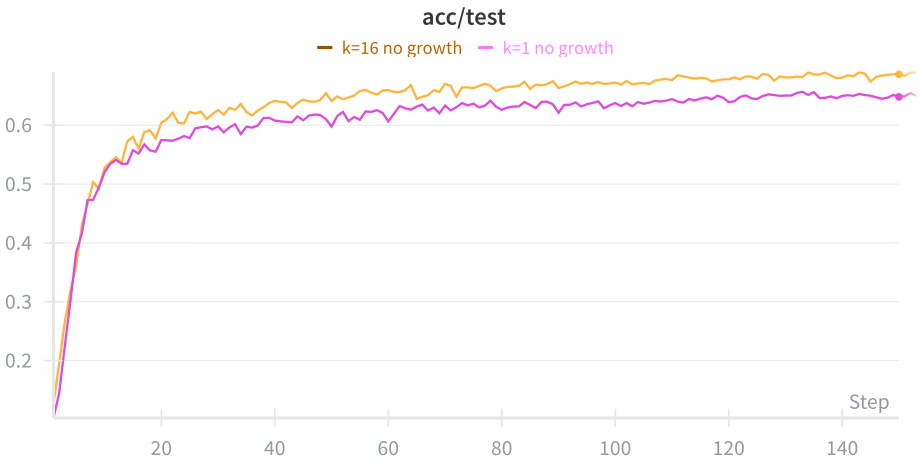

Figure 7: Test accuracy on Imagenette over 150 epochs, with MLP hidden size 1, no growth, 3 blocks with 3 heads each, $k = 16$ VS $k = 1$.

6.3 Comparing a growth training against a no-growth training

On Figure 8, Figure 9 and Figure 10, we can see the results of an experiment on CIFAR100 comparing growth and no growth. For the growth, we initialize the each head with $k = 1$, and let the model slowly build up it's k dimensions. The growth accuracy starts small but seem to slowly converge towards the no-growth accuracy. Note: The train accuracy is lower than the two others because we used data augmentations on the train set.

Unfortunately, we didn't test our results yet with bigger/more complex models, furthermore, for now we only tested the "One-shot" approach, we didn't test the "Two-shots" approach yet.

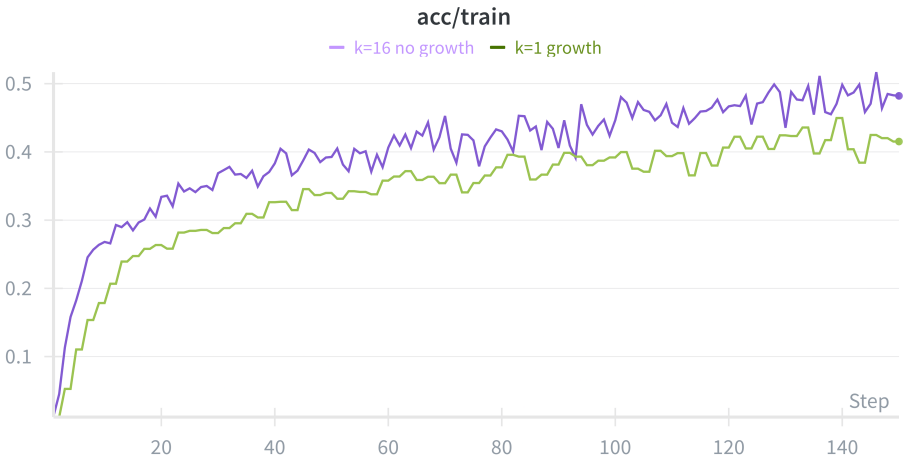

Figure 8: Train accuracy on CIFAR100 over 150 epochs, with $d_e = 64$ hidden MLP size $W_V W_O : 16, W_\xi W_\zeta : 512$, 3 blocks with 2 heads each, $k = 16$ no growth VS initial $k = 1$ with growth.

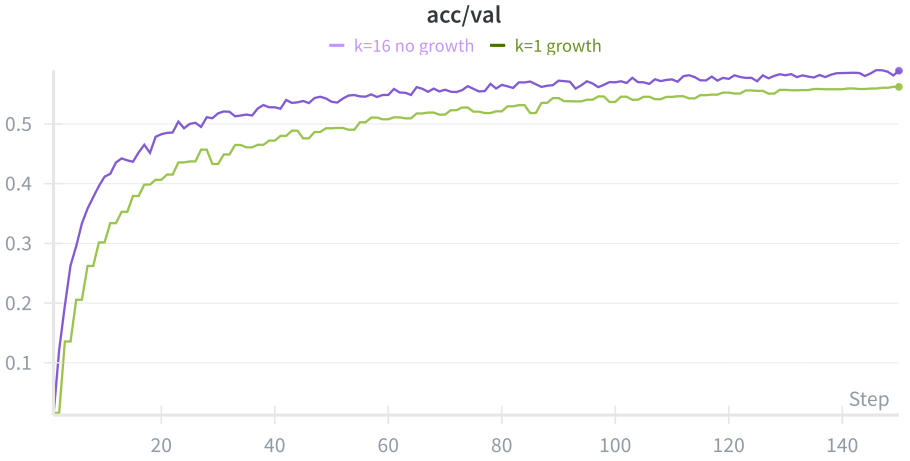

Figure 9: Validation accuracy, same parameters as Figure 8

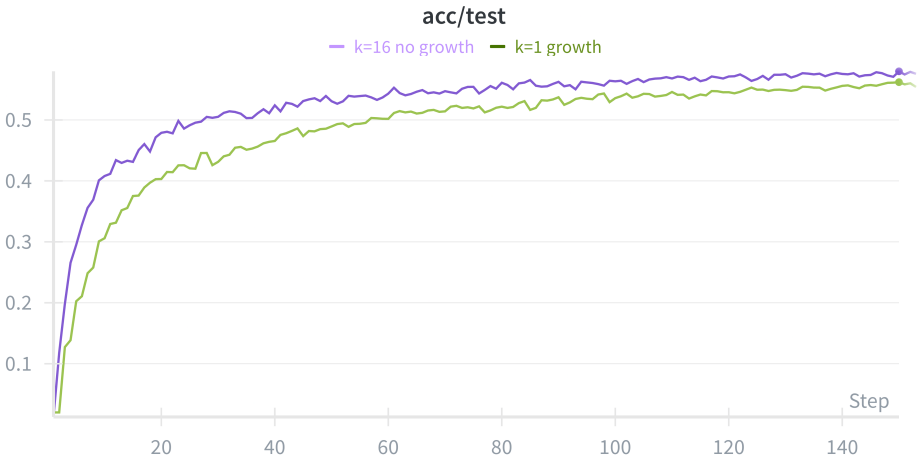

Figure 10: Test accuracy, same parameters as Figure 8

7 Conclusion

We developed closed-form update rules to grow attention heads by projecting a functional target onto the tangent space induced by (W_Q, W_K), and we showed how the resulting system can be solved stably. In controlled experiments, the observed behavior (residuals, alignment, and effect of SVD-based growth) generally matches the theoretical predictions, which suggests the derivations are sound.

That said, the practical cost remains non-trivial. Even with statistic reuse, the Kronecker structures and vectorized solves can be heavy in memory and time at today's model scales. We suspect that scalable variants like iterative solvers, preconditioning, and careful line-search scaling—will be needed before this is scalable in larger settings.

Finally, our empirical scope is modest. Small benchmarks are useful for debugging, but they are limited in their capacity of judging whether growing k consistently helps Transformers. A fair assessment will require larger, more complex datasets and stronger training recipes where ViTs typically benefit. In short: the approach appears principled and promising, the next step is to scale the experiments, thoroughly test in experimental setting all the mathematical theory, and see whether the gains persist where they matter.

8 Future work

- Find an iterative algorithm for both parts of the "Two-shots" approach, to ensure better scalability.
- The experiments seem to have suffered from a lack of complexity, we can try to experiment with more complex models and datasets, to allow the k dimension to be more significant in the loss reduction.
- Transformers present two MLP structures, $W_\xi W_\zeta$ and $W_V W_O$. As the TAU team developed a growing MLP algorithm, we can try to combine both to have a more "fully growable" transformer. We did work towards this, but problems about comparing the different growth criteria remains.
- Integrate the growable Transformer code inside Gromo, the growing neural networks library developed by the TAU team (growingnet, 2025).

Appendix

A.1 Vectorization implementation

A.1.1 Avoid forming $K_{e,k}$

Here is how we can get P:

```
def K_perm_indices(e, k):
    idx = torch.arange(e*k)
    return (idx // e) + (idx % e) * k
P = K_perm_indices(e, k)  # shape: (ek,)
M = N[:,P]
```

A.1.2 Applying the usual vec(.) and unvec(.)

```
def vecF(M):  # column-major vectorization
    return M.t().contiguous().reshape(-1)
def unvecF(v, rows, cols):
    return v.reshape(cols, rows).t().contiguous()
```

Bibliography

Allen-Zhu, Z., Li, Y., & Liang, Y. (2019). Learning and Generalization in Overparameterized Neural Networks, Going Beyond Two Layers. Advances in Neural Information Processing Systems, 32, 6158–6169.

Arora, S., Ge, R., Neyshabur, B., & Zhang, Y. (2018). Stronger generalization bounds for deep nets via a compression approach. Proceedings of the 35th International Conference on Machine Learning, 254–263.

Boyd, S., & Vandenberghe, L. (2004). Convex Optimization. Cambridge University Press.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D., Wu, J., Winter, C., ... Amodei, D. (2020). Language Models are Few-Shot Learners. Advances in Neural Information Processing Systems (Neurips) 33, 1877–1901.

Chen, T., Goodfellow, I., & Shlens, J. (2016,). Net2Net: Accelerating Learning via Knowledge Transfer. International Conference on Learning Representations.

Cortes, C., Gonzalvo, J., Kuznetsov, V., Mohri, M., & Yang, S. (2017). AdaNet: Adaptive Structural Learning of Artificial Neural Networks. Proceedings of the 34th International Conference on Machine Learning, 70, 874–883.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., & Houlsby, N. (2021,). An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. International Conference on Learning Representations.

Eckart, C., & Young, G. (1936). The approximation of one matrix by another of lower rank. Psychometrika, 1, 211–218. https://doi.org/10.1007/BF02288367

Frankle, J., & Carbin, M. (2019,). The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks. International Conference on Learning Representations.

growingnet. (2025,). Gromo: Growing networks module for PyTorch.

Han, S., Mao, H., & Dally, W. J. (2016,). Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. International Conference on Learning Representations.

Hendrycks, D., & Gimpel, K. (2023,). Gaussian Error Linear Units (GELUs). https://arxiv.org/abs/1606.08415

Hinton, G., Vinyals, O., & Dean, J. (2015). Distilling the knowledge in a neural network. Arxiv Preprint Arxiv:1503.02531.

Howard, J. (2019, March). Imagenette: A smaller subset of 10 easily classified classes from Imagenet. GitHub. https://github.com/fastai/imagenette

Krizhevsky, A., Nair, V., & Hinton, G. CIFAR-100 (Canadian Institute for Advanced Research). http://www.cs.toronto.edu/~kriz/cifar.html

Krizhevsky, A., Nair, V., & Hinton, G. (2009,). CIFAR-10 Dataset.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. Nature, 521(7553), 436–444. https://doi.org/10.1038/nature14539

Magnus, J. R., & Neudecker, H. (1979). The Commutation Matrix: Some Properties and Applications. The Annals of Statistics, 7(2), 381–394. https://doi.org/10.1214/aos/1176344621

Maile, K., Rachelson, E., Luga, H., & Wilson, D. G. (2022). When, where, and how to add new neurons to ANNs. Proceedings of the Automl Workshop at ICML, 188, 1–13.

Martens, J. (2020). New Insights and Perspectives on the Natural Gradient Method. Journal of Machine Learning Research, 21(146), 1–76. https://jmlr.org/papers/v21/17-678.html

Michel, P., Levy, O., & Neubig, G. (2019). Are Sixteen Heads Really Better than One?. Advances in Neural Information Processing Systems, 32, 14014–14024.

Mirsky, L. (1960). Symmetric gauge functions and unitarily invariant norms. The Quarterly Journal of Mathematics, 11, 50–59. https://doi.org/10.1093/qmath/11.1.50

Neyshabur, B., Salakhutdinov, R., & Srebro, N. (2015,). Path-SGD: Path-Normalized Optimization in Deep Neural Networks. https://arxiv.org/abs/1506.02617

Padlewski, P., & Djolonga, J. (2023, March 31). Scaling Vision Transformers to 22 Billion Parameters. https://research.google/blog/scaling-vision-transformers-to-22-billion-parameters

Petersen, K. B., & Pedersen, M. S. (2012, November). The Matrix Cookbook. https://www2.compute.dtu.dk/pubdb/views/edoc_download.php/3274/pdf/imm3274.pdf

Real, E., Aggarwal, S., Huang, Y., & Le, Q. V. (2017). Large-Scale Evolution of Image Classifiers. Proceedings of the 34th International Conference on Machine Learning, 70, 2902–2911.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Driessche, G. van den, Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., & Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. Nature, 529(7587), 484–489. https://doi.org/10.1038/nature16961

Touvron, H., Cord, M., Douze, M., Massa, F., Sablayrolles, A., & Jégou, H. (2021). Training data-efficient image transformers & distillation through attention. Proceedings of the 38th International Conference on Machine Learning, 139, 10347–10357.

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., & Babaei, Y. *. a. (2023). Llama 2: Open foundation and fine-tuned chat models. Arxiv Preprint Arxiv:2307.09288.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, {., & Polosukhin, I. (2017). Attention Is All You Need. Advances in Neural Information Processing Systems, 30, 5998–6008.

Verbockhaven, M., Chevallier, S., Charpiat, G., & Rudkiewicz, T. (2024,). Growing Tiny Networks: Spotting Expressivity Bottlenecks and Fixing Them Optimally. https://arxiv.org/abs/2405.19816

Zoph, B., & Le, Q. V. (2017). Neural Architecture Search with Reinforcement Learning. Arxiv Preprint Arxiv:1611.01578.