

Retrieval-Centric Deep Learning in Growing Nonparametric Neural Networks

Maximilian Schlegel^{*1} Rajai Nasser^{*1} Seijin Kobayashi^{*1} Yanick Schimpf^{*1}
 Oliver Sieberling^{*2} Robert Obryk^{*1}

Kazuki Irie^{*3} João Sacramento^{*1} Johannes von Oswald^{*1}

¹Google, Paradigms of Intelligence Team, Zurich, Switzerland

²MIT CSAIL, Cambridge, MA, USA

³Department of Computer Science and Wu Tsai Institute
 Yale University, New Haven, CT, USA

Abstract

We investigate a general-purpose layer for deep learning that, instead of compressing arbitrary-size training data into fixed-size weight matrices, stores a new pair of key-value representations for every data point during training, and retrieves and recombines these representations through an attention mechanism at inference time—resulting in a growing neural net (NN). While Irie et al. [1] have put forward this perspective from the classic duality expressing any linear layer in a deep NN trained by gradient descent as linear attention (LA) over the training data points, replacing LA by more powerful attention functions, as they suggest, turns out to be non-trivial: we show that naively applying learning rules from the LA case to advanced kernels does not lead to principled optimization. Here we fill this gap and develop functional gradient-based learning rules for kernelized attention layers, based on radial basis function (RBF) and softmax-like kernels—establishing the principled “retrieval-centric deep learning” (RCDL) paradigm. Empirically, we demonstrate the promising performance and learning-efficiency of RCDL on image classification and synthetic teacher-student learning tasks. Moreover, we show that replacing LA in the dual form of NNs by advanced LA variants, namely MesaNet/DeltaNet, yields a formal connection to recently proposed optimizers for conventional fixed-size NNs, offering a novel perspective on deep learning optimization.

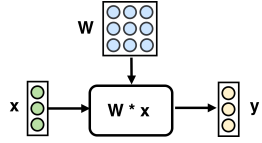
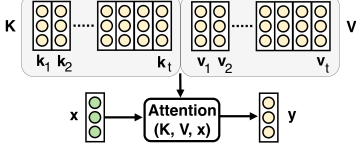
1 Introduction

Traditional deep learning uses gradient descent to compress arbitrary-size training datasets and/or experiences into fixed-size weight matrices of a deep neural network (NN). A classic but lesser-known mathematical property thereof is that any linear layer in such a gradient descent-trained deep NN admits a nonparametric dual form: Its exact input/output relation can also be achieved by an equivalent “key-value memory system” that stores all the training datapoints presented to the layer during training and the initial weights, and produces outputs through *retrieval* using “unnormalized dot-product attention” over the entire training experience stored in the form of “key/value pairs” [2, 1] (reviewed in Sec. 2).

From a modern sequence modeling research perspective, however, such an unnormalized dot-product attention represents a form of “linear attention” [3, 4], which is well-known to be sub-optimal to process sequences, e.g., compared with the softmax attention [5] commonly used in the transformer [6].

^{*}All authors contributed equally. Correspondence: {schlegelm, rajainasser, jvoswald}@google.com, osieberl@mit.edu, kazuki.irie@yale.edu. Public code: <https://github.com/paradigms-of-intelligence/rcdl>

Table 1: Comparison of parametric vs. retrieval-centric deep learning paradigms.

Model	<i>Parametric neural networks</i>	<i>Growing nonparametric networks</i>
Building block	Linear layer	Attention layer
Learnable state (for t train examples)	Fixed-size weight matrix $\mathbf{W} \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$	Growing database of keys & values $\mathcal{D} = \{\mathbf{K} \in \mathbb{R}^{d_{\text{in}} \times t}, \mathbf{V} \in \mathbb{R}^{d_{\text{out}} \times t}\}$
Inference ($\mathbf{x} \in \mathbb{R}^{d_{\text{in}}}, \mathbf{y} \in \mathbb{R}^{d_{\text{out}}}$)	$\mathbf{y} = \mathbf{W}\mathbf{x}$	$\mathbf{y} = s_{\mathcal{D}}(\mathbf{x}) = \text{Attention}(\mathbf{K}, \mathbf{V}, \mathbf{x})$
Illustration		
Forming deep models	Compose linear layers and nonlinearities, e.g., $\sigma(\mathbf{W}_2\sigma(\mathbf{W}_1\mathbf{x}))$	Compose attention layers and nonlinearities, e.g., $\sigma(s_{\mathcal{D}_2}(\sigma(s_{\mathcal{D}_1}(\mathbf{x}))))$
Learning objective	Minimize a global loss $\mathcal{L}$ (e.g., via gradient descent)	
Learning update (e.g., gradient-based)	Update the fixed weight matrix (e.g., $\Delta\mathbf{W} \propto -\nabla_{\mathbf{W}}\mathcal{L} = -\nabla_{\mathbf{y}_t}\mathcal{L}\mathbf{x}_t^\top$)	Append a new key-value pair (e.g., $\mathbf{k}_t = \mathbf{x}_t, \mathbf{v}_t \propto -\nabla_{\mathbf{y}_t}\mathcal{L}$)
Key question	How to update the weight matrix?	What key-value vectors to append?

This observation has raised a natural question: can we achieve more powerful deep learning systems by replacing the linear attention (LA) underlying deep NNs by a more powerful attention function [1, 7]? While this question has been left unanswered, the time is ripe to revisit it, especially given how recent research in sequence modeling has advanced LA and provided insights to interpret and improve LA through various angles, including its connection to fast weight programming and learning rules [8–15], the underlying local optimization perspectives [16–23], as well as traditional sequence processing tools such as gating [24, 25].

Here we pursue two approaches that aim to go beyond current deep learning by extending the LA mechanism underlying any modern deep NNs’ dual form. First, we advance the retrieval-centric view of the dual form by developing a theory for the “growing neural network” obtained by introducing a (non-linearizable) kernel to the dual form’s LA, and devise its learning algorithms (which prescribe the key and value representations to be stored for each data point) that optimize a given global objective function. We study the cases of positive definite kernels (e.g., the classic radial basis function (RBF) [26]) and the “weighted softmax” kernel we propose. In fact, while Irie et al. [1] have suggested to directly “insert” the softmax function in the corresponding linear attention to enhance retrieval from NNs’ training experience, devising a principled learning algorithm for such a system turns out to be highly non-trivial, as such an insertion invalidates the optimality of the key and value representations computed in the linear case. This is especially true for the softmax, for which we introduce a closely related “weighted softmax kernel” to formally derive a learning rule for the resulting retrieval-centric growing NN. We conduct experiments using classic image classification datasets and teacher-student learning tasks, and demonstrate promising results in performance and sample efficiency compared to the standard deep NNs with fixed-size weight matrices.

The second extension we pursue is to leverage recent advances in linear attention, namely the weight updates of DeltaNet [9, 14], and MesaNet [18]. In contrast to the first direction above, this approach gives rise to optimizers that improve conventional weight-based learning. We derive a formal connection to a recently proposed optimizer and second-order optimization, and empirically demonstrate the efficiency of the resulting optimizers, with competitive performance to Muon [27].

Finally, as a new framework that represents a departure from standard deep learning (see comparison in Table 1), retrieval-centric deep learning (RCDL) still faces many challenges. Perhaps the most central one is that model size grows at every training step when a nonlinear kernel is chosen. We finish our paper by presenting first results on a hybrid learning algorithm which manages capacity allocation dynamically, by combining standard (parametric) gradient updates with model growing (non-parametric) steps, triggered only when the inputs to a given layer are sufficiently distinct from past ones. We show empirically that this hybrid algorithm is able to match the performance of the non-parametric learner at a small fraction of its size.

2 Background: Dual Form of a Linear Layer

Here we briefly review the dual form of a linear layer trained by gradient descent as a linear attention system [2, 1], which is our starting point. Let $t, T, d_{\text{in}}, d_{\text{out}}$ denote positive integers. Consider a linear layer inside an arbitrary complex/deep neural network, that transforms a layer input $\mathbf{x} \in \mathbb{R}^{d_{\text{in}}}$ into a layer output $\mathbf{y} \in \mathbb{R}^{d_{\text{out}}}$ as $\mathbf{y} = \mathbf{W}\mathbf{x}$ where $\mathbf{W} \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$ is a weight matrix.

Given an arbitrary differentiable loss function $\mathcal{L}$ and a sequence $(\mathbf{x}_1, \dots, \mathbf{x}_T)$ of T training inputs presented to the layer, with $\mathbf{x}_t \in \mathbb{R}^{d_{\text{in}}}$ for $1 \leq t \leq T$, we denote the corresponding (backpropagated) error signals as $(\mathbf{v}_1, \dots, \mathbf{v}_T)$ with $\mathbf{v}_t \in \mathbb{R}^{d_{\text{out}}}$ defined by gradient descent, that is $\mathbf{v}_t = -\eta_t \nabla_{\mathbf{y}_t} \mathcal{L}$ where $\eta_t \in \mathbb{R}_{>0}$ is the learning rate and $\nabla_{\mathbf{y}_t} \mathcal{L}$ is the gradient of the loss w.r.t. the layer output $\mathbf{y}_t$ corresponding to input $\mathbf{x}_t$.

At training step t , gradient descent updates the weight matrix (WM), initialized with $\mathbf{W}_0$, as $\mathbf{W}_t = \mathbf{W}_{t-1} - \eta_t (\nabla_{\mathbf{y}_t} \mathcal{L}) \otimes \mathbf{x}_t = \mathbf{W}_{t-1} + \mathbf{v}_t \otimes \mathbf{x}_t$, where $\otimes$ denotes the outer product, that is $\mathbf{v}_t \otimes \mathbf{x}_t = \mathbf{v}_t \mathbf{x}_t^\top$. We can unfold this equation to express the WM at step T as: $\mathbf{W}_T = \mathbf{W}_0 + \sum_{t=1}^T \mathbf{v}_t \mathbf{x}_t^\top$.

Thus, the layer output for an arbitrary input $\mathbf{q} \in \mathbb{R}^{d_{\text{in}}}$ can be expressed as:

$$\mathbf{y} = \mathbf{W}_T \mathbf{q} = \mathbf{W}_0 \mathbf{q} + \sum_{t=1}^T \mathbf{v}_t \mathbf{x}_t^\top \mathbf{q} = \mathbf{W}_0 \mathbf{q} + \sum_{t=1}^T \mathbf{v}_t \phi(\mathbf{x}_t, \mathbf{q}) \quad (1)$$

where the last term is a weighted sum of vectors $\mathbf{v}_t \in \mathbb{R}^{d_{\text{out}}}$ with weights $\mathbf{x}_t^\top \mathbf{q} \in \mathbb{R}$, which can be interpreted as an (unnormalized) ‘‘attention mechanism’’ by identifying $\mathbf{v}_t$ as values, $\mathbf{x}_t$ as keys, and $\mathbf{q}$ as the query; and $\phi(\mathbf{x}_t, \mathbf{q}) = \mathbf{x}_t^\top \mathbf{q}$ as attention weights [28, 6, 5]. To be more specific, this is (a simple form of) linear attention [3, 4].

This implies that the input/output relation of the linear layer can also be realized by a ‘‘growing nonparametric neural network’’: Instead of storing the fixed-size weight matrix $\mathbf{W}$ as the system’s training memory, the system maintains a growing database by storing a new pair $(\mathbf{k}_t, \mathbf{v}_t)$ of key-value vectors for each training data point—the layer input $\mathbf{k}_t = \mathbf{x}_t \in \mathbb{R}^{d_{\text{in}}}$ as the ‘‘key’’ and the corresponding error signal $\mathbf{v}_t = -\eta_t \nabla_{\mathbf{y}_t} \mathcal{L} \in \mathbb{R}^{d_{\text{out}}}$ as the ‘‘value’’ vector (in addition to the initial weight matrix $\mathbf{W}_0$). Inference is performed through the linear attention mechanism above (Eq. 1).

Given the ubiquity of linear layers in deep neural networks, this dual view suggests that a fundamental mechanism linking training and inference in virtually any deep learning systems is based on (a cascade of) this simple linear attention mechanism. However, recent research on linear attention in sequence modeling indicates that such ‘‘simple linear attention’’ mechanisms are highly suboptimal. This points to the potential for fundamentally improving deep learning by enhancing the underlying linear attention mechanism.

3 Growing Nets with Non-Linear Attention

With the goal of developing a general-purpose deep learning layer that can improve upon the conventional linear layer, we replace ϕ in Eq. 1 by a more powerful kernel that is not reducible to a finite-dimensional linear map. Consequently, the resulting layer can no longer be converted into a linear layer with a fixed-size weight matrix. Instead, we obtain a ‘‘growing layer’’ in which a new pair of key/value representations is added to the system for each training datapoint. This raises a fundamental question: how should these key and value representations be chosen to implement a *principled* learning algorithm that minimizes a given loss function? Here we formally derive such algorithms in two classes of powerful attention functions.

3.1 Positive Definite Kernel Attention

We first investigate the case where ϕ in Eq. 1 is a positive definite kernel (PDK). By the Moore-Aronszajn theorem [29], any PDK admits a representation as an inner product of feature embeddings into a Hilbert space $\mathcal{H}_\phi$. That is, there exists a map $\psi_\phi : \mathbb{R}^{d_{\text{in}}} \rightarrow \mathcal{H}_\phi$ such that $\phi(\mathbf{x}, \mathbf{x}') = \langle \psi_\phi(\mathbf{x}), \psi_\phi(\mathbf{x}') \rangle_{\mathcal{H}_\phi}$.

This allows us to view the nonparametric layer as a linear layer operating on the implicit feature space $\mathcal{H}_\phi$ with an infinite-dimensional weight matrix $\mathbf{W} \in \mathbb{R}^{d_{\text{out}} \times \infty}$. In Sec. A, we rigorously

show that performing standard gradient descent on this implicit weight matrix is mathematically equivalent to appending a new key-value pair to the database at every step. Analogously to the analysis of the linear kernel case in Sec. 2, the gradient update of the implicit weight matrix is $\Delta \mathbf{W} \propto (-\eta_t \nabla_{\mathbf{y}_t} \mathcal{L}) \psi_\phi(\mathbf{x}_t)^\top$. This translates exactly to adding a new term $(-\eta_t \nabla_{\mathbf{y}_t} \mathcal{L}) \phi(\mathbf{x}_t, \cdot)$ to the layer’s output function.

Consequently, to build a growing layer based on a PDK, we retain the exact key-value definitions from the linear case: setting the key $\mathbf{k}_t = \mathbf{x}_t$ and the value $\mathbf{v}_t = -\eta_t \nabla_{\mathbf{y}_t} \mathcal{L}$ implements a principled loss-minimizing learning rule in the RKHS. In our experiments, we use the radial basis function (RBF) kernel—a classic PDK [26].

3.2 Softmax and Weighted Softmax Attention

While PDKs offer a robust theoretical foundation, modern deep learning relies heavily on softmax attention, which empirically outperforms unnormalized kernel attention in many sequence modeling tasks [6]. Here we study the case where we replace ϕ in Eq. 1 by the softmax function (more formally, $\phi(\mathbf{x}_t, \mathbf{q})$ becomes $\phi(\mathbf{x}_t, \mathbf{q}; \{\mathbf{x}_\tau\}_{1 \leq \tau \leq T})$ as it involves normalization). As it turns out, theoretically dealing with the standard softmax attention is highly non-trivial (as we discuss below; and in Appendix B.3). Instead, we introduce a “weighted softmax kernel” and derive a learning algorithm for the corresponding growing NN.

Let $\mathcal{D} \subset \mathbb{R}^{d_{\text{in}}} \times \mathbb{R}^{d_{\text{out}}}$ be a finite set of key-value pairs or “database”. This database induces the standard softmax attention function $s_{\mathcal{D}} : \mathbb{R}^{d_{\text{in}}} \rightarrow \mathbb{R}^{d_{\text{out}}}$ defined as: $s_{\mathcal{D}}(\mathbf{q}) = \sum_{(\mathbf{k}, \mathbf{v}) \in \mathcal{D}} e^{\langle \mathbf{k}, \mathbf{q} \rangle} \mathbf{v} / \sum_{(\mathbf{k}, \mathbf{v}) \in \mathcal{D}} e^{\langle \mathbf{k}, \mathbf{q} \rangle}$.

Consider the objective of making a small “corrective” adjustment to $\mathcal{D}$ to move $s_{\mathcal{D}}$ in a direction that minimizes a global loss. Ideally, we would achieve this by adding a new key-value pair to the database, analogous to the “growing layers” derived in Sec. 3.1.

However, this is not possible in an infinitesimal manner for this softmax attention. Introducing any new key *discontinuously* alters the denominator, regardless of how small the associated value vector is.

To enable the “smooth addition” of new key-value pairs, we introduce a scalar weight for each entry. This leads to the “weighted softmax attention” formulation, where the database $\mathcal{D}$ consists of a finite set of triples $(w, \mathbf{k}, \mathbf{v}) \in \mathbb{R}^+ \times \mathbb{R}^{d_{\text{in}}} \times \mathbb{R}^{d_{\text{out}}}$, and the retrieval function becomes:

$$s_{\mathcal{D}}(\mathbf{q}) = \frac{\sum_{(w, \mathbf{k}, \mathbf{v}) \in \mathcal{D}} w e^{\langle \mathbf{k}, \mathbf{q} \rangle} \mathbf{v}}{\sum_{(w, \mathbf{k}, \mathbf{v}) \in \mathcal{D}} w e^{\langle \mathbf{k}, \mathbf{q} \rangle}}. \quad (2)$$

In the remainder of this section, we develop a theoretical framework that allows us to derive a “growing database” formulation which corresponds to performing a gradient descent in a space containing all weighted softmax functions.

Now consider an infinite database containing a weight for every possible key-value pair. We formalize this by defining a weight function $w : \mathbb{R}^{d_{\text{in}}} \times \mathbb{R}^{d_{\text{out}}} \rightarrow \mathbb{R}^+$, which induces the continuous softmax function $s_w : \mathbb{R}^{d_{\text{in}}} \rightarrow \mathbb{R}^{d_{\text{out}}}$ as:

$$s_w(\mathbf{q}) = \frac{\int_{\mathbb{R}^{d_{\text{in}}} \times \mathbb{R}^{d_{\text{out}}}} w(\mathbf{k}, \mathbf{v}) e^{\langle \mathbf{k}, \mathbf{q} \rangle} \mathbf{v} d\mathbf{k} d\mathbf{v}}{\int_{\mathbb{R}^{d_{\text{in}}} \times \mathbb{R}^{d_{\text{out}}}} w(\mathbf{k}, \mathbf{v}) e^{\langle \mathbf{k}, \mathbf{q} \rangle} d\mathbf{k} d\mathbf{v}}. \quad (3)$$

Appendix B.1 provides further remarks on the functional space of weighted softmax functions.

The case of Gaussian mixtures. We analyze the case where $w : \mathbb{R}^{d_{\text{in}}} \times \mathbb{R}^{d_{\text{out}}} \rightarrow \mathbb{R}^+$ represents the density of a Gaussian mixture. Let $\mathcal{D} = \{(w_i, \mathbf{k}_i, \mathbf{v}_i) : 1 \leq i \leq n\} \subset \mathbb{R}^+ \times \mathbb{R}^{d_{\text{in}}} \times \mathbb{R}^{d_{\text{out}}}$ be a finite set of n weighted key-value pairs, and let $\sigma_k, \sigma_v > 0$. We define the Gaussian mixture density $w_{\mathcal{D}}$ as follows:

$$w_{\mathcal{D}}(\mathbf{k}, \mathbf{v}) = \sum_{i=1}^n \frac{w_i}{(2\pi)^{\frac{d_{\text{in}}+d_{\text{out}}}{2}} \sigma_k^{d_{\text{in}}} \sigma_v^{d_{\text{out}}}} e^{-\frac{\|\mathbf{k}-\mathbf{k}_i\|^2}{2\sigma_k^2} - \frac{\|\mathbf{v}-\mathbf{v}_i\|^2}{2\sigma_v^2}}.$$

This formulation corresponds to mixing n Gaussian densities centered at $(\mathbf{k}_i, \mathbf{v}_i)_{1 \leq i \leq n}$ with respective mixing weights $(w_i)_{1 \leq i \leq n}$. The covariance matrix Σ for each of the n Gaussian components is a block diagonal matrix given by:

$$\Sigma = \begin{bmatrix} \Sigma_k & 0 \\ 0 & \Sigma_v \end{bmatrix} = \begin{bmatrix} \sigma_k^2 \mathbf{I}_{d_{\text{in}}} & 0 \\ 0 & \sigma_v^2 \mathbf{I}_{d_{\text{out}}} \end{bmatrix}. \quad (4)$$

Now, let us compute the softmax function $s_{w_{\mathcal{D}}}$ induced by this mixture density. Substituting $w_{\mathcal{D}}$ into Eq. 3, we can show that (see the full derivation in Appendix B.2):

$$\begin{aligned} s_{w_{\mathcal{D}}}(\mathbf{q}) &= \frac{\int_{\mathbb{R}^{d_{\text{in}}} \times \mathbb{R}^{d_{\text{out}}}} w_{\mathcal{D}}(\mathbf{k}, \mathbf{v}) e^{\langle \mathbf{k}, \mathbf{q} \rangle} \mathbf{v} d\mathbf{k} d\mathbf{v}}{\int_{\mathbb{R}^{d_{\text{in}}} \times \mathbb{R}^{d_{\text{out}}}} w_{\mathcal{D}}(\mathbf{k}, \mathbf{v}) e^{\langle \mathbf{k}, \mathbf{q} \rangle} d\mathbf{k} d\mathbf{v}} \\ &= \frac{\sum_{i=1}^n w_i e^{\langle \mathbf{k}_i, \mathbf{q} \rangle + \frac{1}{2} \sigma_k^2 \|\mathbf{q}\|^2} \mathbf{v}_i}{\sum_{i=1}^n w_i e^{\langle \mathbf{k}_i, \mathbf{q} \rangle + \frac{1}{2} \sigma_k^2 \|\mathbf{q}\|^2}} = \frac{\sum_{i=1}^n w_i e^{\langle \mathbf{k}_i, \mathbf{q} \rangle} \mathbf{v}_i}{\sum_{i=1}^n w_i e^{\langle \mathbf{k}_i, \mathbf{q} \rangle}} = s_{\mathcal{D}}(\mathbf{q}). \end{aligned}$$

This demonstrates that the continuous softmax function induced by the Gaussian mixture density $w_{\mathcal{D}}$ is exactly equivalent to the discrete weighted softmax function $s_{\mathcal{D}}$ defined by the database $\mathcal{D}$. The shared variance-dependent term $e^{\frac{1}{2} \sigma_k^2 \|\mathbf{q}\|^2}$ cancels out in the numerator and denominator, rendering the prediction independent of the specific variances σ_k and σ_v used in the mixture model. It is worth noting that the same cancellation holds for any general block-diagonal covariance matrix Σ (where $\mathbf{k}$ and $\mathbf{v}$ are independent), regardless of the specific correlations within the key or value dimensions.

The functional space of Gaussian-modulated softmax functions. Now observing that Gaussian mixtures with a fixed covariance matrix Σ are absolutely continuous with respect to the centered Gaussian measure $\mu = \mathcal{N}(\mathbf{0}, \Sigma)$, we can analyze the system using this background measure. We define the Gaussian-modulated softmax function for a relative weight function w as:

$$s_w(\mathbf{q}) = \frac{\int_{\mathbb{R}^{d_{\text{in}}} \times \mathbb{R}^{d_{\text{out}}}} w(\mathbf{k}, \mathbf{v}) e^{\langle \mathbf{k}, \mathbf{q} \rangle} \mathbf{v} d\mu(\mathbf{k}, \mathbf{v})}{\int_{\mathbb{R}^{d_{\text{in}}} \times \mathbb{R}^{d_{\text{out}}}} w(\mathbf{k}, \mathbf{v}) e^{\langle \mathbf{k}, \mathbf{q} \rangle} d\mu(\mathbf{k}, \mathbf{v})}. \quad (5)$$

This formulation relates to the original Lebesgue-density formulation via a Radon-Nikodym derivative. If the physical density of the database is $p(\mathbf{k}, \mathbf{v})$, the corresponding functional weight is $w(\mathbf{k}, \mathbf{v}) = \frac{dp}{d\mu}(\mathbf{k}, \mathbf{v})$.

This perspective reveals a crucial structural property: a Gaussian mixture in the physical space corresponds to a linear combination of exponential functions in the functional space. Specifically, the ratio of a shifted Gaussian $\mathcal{N}(\mathbf{x}|\mathbf{u}, \Sigma)$ to the base measure $\mathcal{N}(\mathbf{x}|\mathbf{0}, \Sigma)$ is proportional to $e^{\langle \mathbf{x}, \Sigma^{-1} \mathbf{u} \rangle}$. Consequently, adding a key-value pair to the database is mathematically equivalent to adding an exponential basis function to the weight $w(\mathbf{k}, \mathbf{v})$. This justifies searching for updates in the span of exponential functions during gradient descent.

Let $\mathcal{H}_{\sigma_k, \sigma_v}$ be the space of functions $w : \mathbb{R}^{d_{\text{in}}} \times \mathbb{R}^{d_{\text{out}}} \rightarrow \mathbb{R}$ such that

$$\mathbb{E}_{(\mathbf{k}, \mathbf{v}) \sim \mathcal{N}(\mathbf{0}, \Sigma)} [w(\mathbf{k}, \mathbf{v})^2 e^{\langle \mathbf{k}, \mathbf{q} \rangle + \langle \mathbf{v}, \mathbf{v}' \rangle}] < \infty, \quad \forall \mathbf{q} \in \mathbb{R}^{d_{\text{in}}}, \forall \mathbf{v}' \in \mathbb{R}^{d_{\text{out}}}.$$

We endow $\mathcal{H}_{\sigma_k, \sigma_v}$ with the $L_2(\mu)$ inner product: $\langle w, w' \rangle_{\mathcal{H}_{\sigma_k, \sigma_v}} = \mathbb{E}_{(\mathbf{k}, \mathbf{v}) \sim \mathcal{N}(\mathbf{0}, \Sigma)} [w(\mathbf{k}, \mathbf{v}) w'(\mathbf{k}, \mathbf{v})]$.

Functional gradient descent update rule. We derive the update rule for the weight function $w \in \mathcal{H}_{\sigma_k, \sigma_v}$ by performing functional gradient descent on a loss function $\mathcal{L}(s_w(\mathbf{q}), \mathbf{y})$. For a single observation $(\mathbf{q}_t, \mathbf{y}_t)$, we compute the Fréchet derivative of the loss functional $J(w) = \mathcal{L}(s_w(\mathbf{q}_t), \mathbf{y}_t)$ and identify the functional gradient $\nabla_w J$ via the Riesz representation theorem in $\mathcal{H}_{\sigma_k, \sigma_v}$. As detailed in Sec. C, the gradient $\nabla_w J$ takes a specific form involving the exponential kernel $e^{\langle \mathbf{k}, \mathbf{q}_t \rangle}$ and the error signal:

$$\nabla_w J(\mathbf{k}, \mathbf{v}) = \frac{e^{\langle \mathbf{k}, \mathbf{q}_t \rangle}}{D(w)} \langle \nabla_{\mathbf{y}} \mathcal{L}, \mathbf{v} - s_w(\mathbf{q}_t) \rangle \text{ with } D(w) = \int_{\mathbb{R}^{d_{\text{in}}} \times \mathbb{R}^{d_{\text{out}}}} w(\mathbf{k}, \mathbf{v}) e^{\langle \mathbf{k}, \mathbf{q}_t \rangle} d\mu(\mathbf{k}, \mathbf{v}). \quad (6)$$

$D(w)$ is the denominator of the softmax functional induced by w .

The functional update $w_{t+1} \leftarrow w_t - \eta \nabla_w J$ corresponds to adding a new term to the weight function, which is constructively equivalent to appending a new key-value tuple to the database (cf. Sec. C.4). Specifically, this establishes the update rule that adds a tuple $(w_{\text{new}}, \mathbf{k}_{\text{new}}, \tilde{\mathbf{v}}_{\text{new}})$ where the new key is a scaled version of the query, $\mathbf{k}_{\text{new}} = \Sigma_k \mathbf{q}_t = \sigma_k^2 \mathbf{q}_t$, and the weight and value are determined by the prediction error and the local density. To ensure numerical stability, we store the weighted value vector $\tilde{\mathbf{v}} = w\mathbf{v}$, leading to the robust update rules:

$$w_{\text{new}} \propto \eta \langle \mathbf{s}_t, \mathbf{g}_t \rangle, \quad \tilde{\mathbf{v}}_{\text{new}} \propto -\eta \Sigma_v \mathbf{g}_t - \eta \sigma_v^2 \mathbf{g}_t, \quad (7)$$

where $\mathbf{g}_t = \nabla_{\hat{\mathbf{y}}} \mathcal{L}$ and $\mathbf{s}_t = s_{w_t}(\mathbf{q}_t)$. Refer to Sec. C for more details. In the case of mini-batch stochastic gradient descent with a batch of size B , the linearity of the gradient operator implies that the total functional gradient is the sum of individual gradients centered at distinct queries. Consequently, a single step corresponds to adding B distinct tuples to the database simultaneously, reflecting the non-parametric nature of the approach where model complexity grows with data.

Remark 3.1. Given that $\mathbf{k}_{\text{new}} = \sigma_k^2 \mathbf{q}_t$, one can interpret σ_k^2 as the inverse temperature of the resulting softmax function. In practice, we can store $\mathbf{k}_{\text{new}} = \mathbf{q}_t$ and instead use a temperature hyperparameter T in the Softmax, as is typically done¹, i.e., the final weighted-softmax function becomes:

$$s_{\mathcal{D}}(\mathbf{q}) = \frac{\sum_{(w, \mathbf{k}, \tilde{\mathbf{v}}) \in \mathcal{D}} e^{\frac{1}{T\sqrt{d_{\text{in}}}} \langle \mathbf{k}, \mathbf{q} \rangle} \tilde{\mathbf{v}}}{\sum_{(w, \mathbf{k}, \tilde{\mathbf{v}}) \in \mathcal{D}} e^{\frac{1}{T\sqrt{d_{\text{in}}}} \langle \mathbf{k}, \mathbf{q} \rangle} w}. \quad (8)$$

In Appendix D, we provide further remarks regarding extensions of the learning algorithm above (corresponding to vanilla gradient descent) with weight decay and momentum.

3.2.1 Handling negative weights: clipping and gauge invariance

One challenge with the update rule of equation 7 is that when $\langle \mathbf{s}_t, \mathbf{g}_t \rangle < 0$ we get a *negative* weight $w_{\text{new}} < 0$. This may cause denominator cancellation and numerical instability (cf. Sec. C.5). A simple heuristic to enforce non-negativity is via clipping, $w_{\text{new}} \leftarrow \max(0, w_{\text{new}})$.

Another method for addressing the negative weights issue without clipping is via a translational symmetry of the softmax functional, which we call *gauge-invariance* (cf. Sec. C.7): If we shift all the value vectors in the database by $+\mathbf{b}$, the output of the weighted softmax also shifts by $+\mathbf{b}$. Hence, if we then subtract $\mathbf{b}$ from the output, we recover the original weighted softmax. Therefore, by equipping the softmax functional with an auxiliary global bias $\mathbf{b}$, the model gains a translational symmetry between the values and the bias (cf. Proposition C.3). This means that the same function has a different representation for each offset (gauge). We can dynamically choose an offset frame—a “virtual gauge” $\mathbf{b}_t$ along the gradient direction—that guarantees that the effective projection $\langle \mathbf{g}_t, \mathbf{s}_t + \mathbf{b}_t \rangle$ is strictly positive, ensuring $w_{\text{new}} > 0$. Refer to Sec. C.7 for a detailed description of this method.

3.3 Sublinear Growth with Hybrid Non-Parametric and Parametric Learning

Appending a new key-value tuple at every training step incurs a linear memory growth $\mathcal{O}(T)$ with respect to the number of training steps T . To achieve scalable, sublinear growth, we introduce a hybrid learning rule that dynamically balances non-parametric expansion with parametric refinement: We treat the existing key-value pairs in the database as learnable parameters optimized via standard optimizers (e.g., Adam or AdamW), while conditionally growing the database using functional gradient descent only when a newly encountered key $\mathbf{x}$ is sufficiently distinct from the existing keys $\mathbf{k}_j$ in the database, based on a threshold parameter τ , i.e., $\min_j \|\mathbf{x} - \mathbf{k}_j\|_2 > \tau$. By restricting memory growth to genuinely novel data while leveraging parametric optimization for representation tuning, this hybrid mechanism dramatically curbs memory footprint while preserving the performance advantages of the fully non-parametric model (see Figure 5).

4 Advanced Linear Attention Perspectives

Sec. 3 focused on improving the LA underlying the dual form of NNs (Eq. 1) by introducing nonlinearity. Here we plug advanced LA models into it, and show how they lead to improved NN optimizers.

We focus on the mesa-layer [17, 18], which replaces the linear attention of Eq. 1 by the best possible linear key-value map, in the sense that it achieves the lowest (regularized) squared error over the keys and values seen up to time t :

$$\mathbf{W}_t = \arg \min_{\mathbf{W}} \frac{1}{2} \sum_{\tau=1}^t \|\mathbf{v}_{\tau} - \mathbf{W} \mathbf{x}_{\tau}\|^2 + \frac{\lambda}{2} \|\mathbf{W}\|_F^2, \quad (9)$$

¹ σ_k^2 relates to the temperature because increasing σ_k^2 scales the positions of new keys quadratically, causing them to spread apart faster than the linear expansion of the standard deviation of individual Gaussian components. Consequently, higher variance σ_k^2 acts as a separation parameter, reducing component overlap and sharpening the attention distribution.

where $\lambda > 0$ is a static (scalar) hyperparameter which controls the strength of regularization, given by the squared Frobenius norm $\|\mathbf{W}\|_F^2$ of the weights. Note that the mesa-layer considers the cumulative (in time) loss, and fully minimizes it. von Oswald et al. [18] empirically found that the mesa-layer outperforms both standard LA and the DeltaNet on various sequence modeling problems.

Interestingly, as we show next, the dual view of a linear layer trained by gradient descent yields an approximate second-order optimizer once we replace linear attention (Eq. 1) by the mesa-layer (Eq. 9). The only remaining step is to determine what the values $\mathbf{v}_t$ should be. In this section, we set them to be the gradient-corrected layer outputs:

$$\mathbf{v}_t = \mathbf{W}_{t-1}\mathbf{x}_t - \eta_t \nabla_{\mathbf{y}_t} \mathcal{L}. \quad (10)$$

An important property of the objective of the mesa-layer (Eq. 9) is that it can be solved analytically. We use this property to obtain our optimization algorithm, which we develop for the mini-batch setting—the current standard in deep learning. Before proceeding, we introduce additional notation to handle mini-batches. We denote by T the number of batches seen so far, each containing B data points. In this setting, $\mathbf{X}_T$ is the $d_{\text{in}} \times B$ matrix that holds the current batch inputs in its columns, and $\mathbf{V}_T$ is the $d_{\text{out}} \times B$ matrix that holds the corresponding gradient-corrected outputs in its columns, following Eq. 10. Additionally, we define $\mathbf{X}_{<T}$ and $\mathbf{V}_{<T}$ as the history of all inputs and gradient-corrected outputs (resp.) observed thus far, excluding the current batch. Solving Eq. 9 analytically, we obtain a closed-form expression for the weights at iteration T i.e., $\mathbf{W}_T = (\mathbf{V}_{<T}\mathbf{X}_{<T}^\top + \mathbf{V}_T\mathbf{X}_T^\top) \cdot (\mathbf{X}_{<T}\mathbf{X}_{<T}^\top + \mathbf{X}_T\mathbf{X}_T^\top + \lambda\mathbf{I})^{-1}$. It turns out that the resulting layer implements exactly a recently proposed optimizer [FOOF; 30], which has been shown to be highly efficient in deep learning. FOOF was derived as a simplification of KFAC [31], itself an approximate second-order optimization method. Intuitively, the optimization speed-up that FOOF (here, the mesa-layer) enjoys stems from the additional inverse matrix which is not present in linear attention. This additional factor applies a whitening transform to the inputs, which yields a full (second-order) Newton update for a shallow linear model, and an approximate second-order update for a deep neural network. We refer to Benzing [30] for an analysis of this optimizer, and to Appendix E for additional details.

As an additional reference, we further consider the DeltaNet [9] case. As derived from the delta rule [32], this case corresponds to performing an online gradient descent step w.r.t. $\mathbf{W}$ on the layer-local, time-dependent loss penalizing the discrepancy between the prediction $\mathbf{W}\mathbf{x}_t$ and a target value $\mathbf{v}_t = -\eta \nabla_{\mathbf{y}_t} \mathcal{L}$:

$$\ell_t(\mathbf{W}) := \frac{1}{2} \|\mathbf{W}\mathbf{x}_t - \mathbf{v}_t\|_2^2 = \frac{1}{2} \|\mathbf{W}\mathbf{x}_t + \eta \nabla_{\mathbf{y}_t} \mathcal{L}\|_2^2 \quad (11)$$

with learning rate λ (corresponding to an online version of the mesa-layer case). This yields the update $\mathbf{W}_t = \mathbf{W}_{t-1} + \lambda(-\eta \nabla_{\mathbf{y}_t} \mathcal{L} - \mathbf{W}_{t-1}\mathbf{x}_t)\mathbf{x}_t^\top$. Appendix E provides additional details.

5 Experiments

Here we conduct experiments to empirically validate theories arising from the two extended views of linear layers above: the newly derived growing neural networks and their learning rules (Sec. 3) and advanced optimizers (Sec. 4).

Basic Settings. The baseline architecture is a residual MLP similar to the transformer’s MLP block [6]; the model has multiple layers with a block structure, where every block of the model updates the residual stream $\mathbf{e} \in \mathbb{R}^{d_e}$ following

$$\mathbf{e} \leftarrow \mathbf{e} + \mathbf{W}_2 \sigma(\mathbf{W}_1 \text{RMSNorm}(\mathbf{e})), \quad (12)$$

with nonlinearity $\sigma(\cdot)$ and RMSNorm prenormalization [33]. The first block is applied to the net input $\mathbf{x} \in \mathbb{R}^{d_x}$. The output layer is the standard softmax layer for classification or a linear layer for regression, using the cross-entropy or mean-squared error loss, respectively.

To find a strong baseline, we perform intensive search over the choice of activation $\sigma(\cdot)$, namely: ReLU [34, 35], GELU [36] and softmax [37]; as well as over the space of optimizers, including plain SGD, Adam [38], AdamW [39], RMSProp [40], and Muon [27]. These serve as the baselines for the MesaNet/DeltaNet-derived optimizers introduced in Sec. 4. We perform a fine-grained scan over hyperparameters, including model depth and nonlinearity type. All our results are averaged over 5 seeds, and shaded areas reflect stdev. unless noted otherwise (see Appendix F.1 for more details). Our nonlinear attention-based growing nets (Sec. 3) follow the same residual architecture, but perform:

$$\mathbf{e} \leftarrow \mathbf{e} + g_{\mathcal{D}}(\text{RMSNorm}(\mathbf{e})), \quad (13)$$

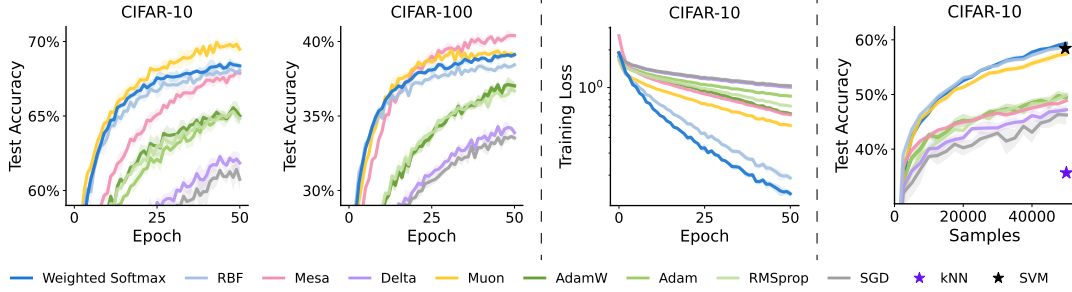


Figure 2: (Left Plots) Test set accuracy on CIFAR-10/100: Growing Weighted Softmax & RBF, while based on plain SGD, significantly outperform the SGD-trained MLP and Adam/RMSProp and are competitive with MLPs trained with Muon. The Mesa/FOOF optimizer achieves high accuracy on CIFAR-10 and performs best on CIFAR-100. (Middle Right) Training loss for Growing Weighted Softmax & RBF decreases much faster than that of the other models. (Right) Test set accuracy on CIFAR-10 after a *single pass* on the training data. Growing Weighted Softmax & RBF are the most efficient learning algorithms. To place our results, we include the test accuracy of two classical (not single pass) methods: k-nearest-neighbor (kNN) and SVM (starred; details in Appendix F.10).

where $g_D(\cdot)$ can either be the weighted-softmax layer (Eq. 2) or the RBF layer (Eq. 1, with ϕ an RBF).

The final output layer is similar but without the residual connection. Importantly, note that the formal learning rules we derived for these growing nets correspond to training them using plain SGD (Sec. 3); derivations for more sophisticated optimizers are left for future work.

Teacher-Student Experiments. We begin with experiments on a synthetic regression problem, where scalar target outputs $y^* \in \mathbb{R}$ are generated by a held-out residual MLP teacher $y^*(x)$ from 100-dimensional inputs $x \in \mathbb{R}^{100}$ [41]. The teacher NN has 1 hidden layer with ReLU non-linearity, residual-stream $d_e = 100$, and random, fixed weights sampled from a LeCun normal distribution (fan-in). The teacher then projects the residual stream activation to an output dimension of 1. The student $y(x)$ is trained through online SGD on the expected squared error loss $\mathbb{E}_{x \sim p(x)}[(y^*(x) - y(x))^2]$, by drawing new data samples at every training iteration. The inputs x are generated by first sampling from a standard normal distribution, and then projecting to the unit sphere. Appendix F provides further setup details.

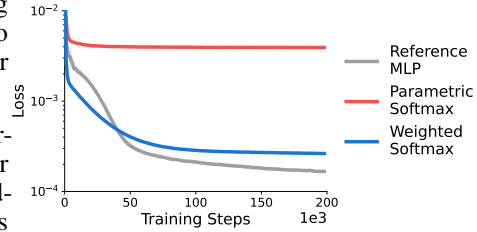


Figure 1: Teacher-student regression loss (mean $\pm$ s.e.m.) comparing: (blue, Weighted Softmax) our growing weighted softmax model; (red, Parametric Softmax) a very wide MLP baseline with width matching the final number of training samples using the softmax activation; and (gray, Reference MLP) an overparameterized student with matched architecture providing a lower bound.

In this task, an overparameterized student—with the same architecture as the teacher, but larger width—is expected to perform very strongly [42]. In Fig. 1 we see that this is indeed the case; the MLP student (denoted by “Reference MLP” in the plot) outperforms all other models, and serves as the gold standard for this task.

We then compare our Growing Weighted Softmax model (GWS; Sec. 3.2) with a fixed-size “parametric softmax” baseline that uses softmax activation (Eq. 12) and dimensions matching the full database size—closely mirroring the GWS model by treating W_1 and W_2 as learnable keys and values [43].

We compare a single-layer GWS with a single-block parametric softmax baseline (with two trainable weight matrices; the best model trained with Adam). We find that GWS significantly outperforms the parametric baseline. Despite the architectural mismatch with the teacher, and despite only having one trainable layer (comprising values and respective weights), our growing weighted softmax model can reach significantly lower loss values.

CIFAR-10/100 Experiments. Next, we investigate the performance of the growing RCDL models on the classic CIFAR-10 and CIFAR-100 image classification tasks [44], using standard data augmentation and non-convolutional architectures. Fig. 2 reports the performance of several parametric baselines and the growing nonparametric models based on RBF and weighted softmax.

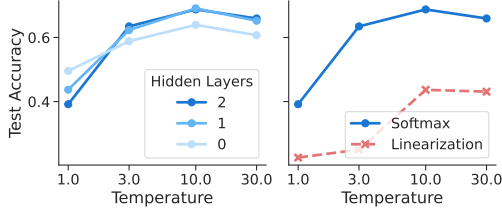


Figure 3: (Left) The growing weighted softmax model trained on CIFAR-10 improves with depth, and displays a non-monotonic temperature behavior w.r.t. test set acc. For the optimal value of $T = 10$ and $d_{\text{in}} = 3072$, $\frac{1}{T\sqrt{d_{\text{in}}}} \langle \mathbf{k}, \mathbf{q} \rangle \in [-5.54, 5.54]$ implying that the softmax layer is operating in the non-linear regime. (Right) This is further confirmed by linearizing post-training the “exp” nonlinearity of the layer, which results in a significant performance degradation.

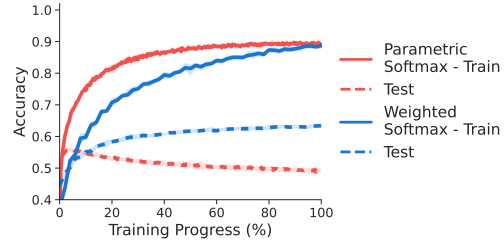


Figure 4: Comparison between parametric softmax (very wide MLP with softmax nonlin.) and our growing non-parametric weighted softmax model. Despite sharing the same architecture, the models behave very differently, both in terms of optimization and generalization: The parametric model displays a widening train-test gap, that is not present in its nonparametric weighted counterpart. We train the former for 2000, the latter for 50 epochs.

Fig. 2/Left shows test accuracy on CIFAR-10/100. We observe that both the growing weighted softmax and RBF models (Sec. 3), whose learning rules are based on plain SGD, significantly outperform the baselines trained by plain SGD, as well as Adam, AdamW and RMSProp; and the GWS model slightly outperforms the RBF variant.

Using more advanced optimizers, the parametric MLP eventually surpasses the SGD-based growing models: the MLP trained by the MesaNet-derived (FOOF) optimizer outperforms all methods on CIFAR-100 and performs on par with the growing variants on CIFAR-10 (see Appendix F.8 for further discussion on the DeltaNet-based one). Similarly, Muon proves to be a strong baseline on both datasets achieving top accuracy on CIFAR-10. These results highlight the importance of developing learning rules for growing models that correspond to these advanced optimizers.

Fig. 2/Middle reports the evolution of training loss on CIFAR-10. Interestingly, the training loss decreases drastically faster for the two nonparametric models compared to all the other methods, while having good generalization performance as shown above; this may be a promising property for tasks that require both good generalization and strong memorization.

Finally, Fig. 2/Right shows the learning efficiency of all methods when trained only on a single pass over non-augmented CIFAR-10 dataset. In this online regime, the growing methods outperform all parametric models including Muon and Mesa. We refer to Sec. G for further experimental results and Sec. G.2 for results using the gauge-invariance version of the GWS layer (described in Sec. 3.2.1).

Analysis. Here we analyze how the GWS model’s test performance varies as a function of the number of layers and its temperature (Sec. 3.2), and verify that the model effectively leverages the nonlinear kernel.

Fig. 3 shows that adding hidden layers improves the performance on CIFAR-10 (though performance saturates with 2 hidden layers), and that there is a non-monotonic dependence on temperature. We also confirm that the model is in the nonlinear regime by ablating the trained model by linearizing the exponential function (i.e., using the approximation $\exp(x) \approx 1 + x$) in the weighted softmax, post-training, which leads to a severe drop in performance.

We further compare the training and generalization behaviors of the GWS model with that of the parametric softmax baseline. Figure 4 shows their training and test accuracy on CIFAR-10 as a function of the training progress. We observe that their behaviors are very different: despite being able to train a huge set of parameters at every iteration, the parametric baseline is slower to fit the training data, despite being highly prone to overfitting; we could not find a hyperparameter configuration that avoided a widening train-test performance gap. The GWS model has no such issues and reaches much higher test accuracy overall.

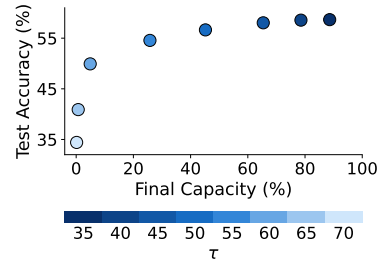


Figure 5: CIFAR-10 performance vs final capacity for various growth thresholds τ . The model retains high test acc. while exhibiting graceful performance degradation when compressing further.

Mixing retrieval-centric and parametric layers. It is also straightforward to mix RCDL layers and parametric layers within the same model. To illustrate this, we conduct a sanity check experiment on vision transformers (ViTs) [45] where we replace the feed-forward block in every ViT layer with an RCDL layer as a drop-in replacement, while keeping all other layers (e.g., self-attention) parametric and training them by gradient descent. Trained for a single epoch on CIFAR-10, the purely parametric ViT achieves 48.4% while our mixed model achieves 54.4% test set accuracy, a promising result opening up many avenues for future research. For more details, see Appendix F.11.

Sub-linear growing. Finally, to mitigate the prohibitive memory and compute growth of vanilla RCDL layers, we provide preliminary results of the hybrid layer described in Sec. 3.3 which combines parametric learning with (conditional) non-parametric growth, again when training CIFAR-10 for one epoch. As shown in Figure 5, the memory growth can be made sublinear while preserving strong performance. Further experimental details can be found in Appendix F.4.

6 Discussion, Related Work, and Conclusion

The RCDL concept has connections to many existing methods in machine learning, such as kernel machines [2, 46, 47, 26] (further related work can be found in Appendix H). In particular, the principled derivation of our update rules—particularly our novel rules for the weighted softmax kernel—closely parallels functional gradient boosting [48, 49]. While this perspective remains relatively underexplored in deep learning, prior work [50] derived an algorithm for constructing networks by iteratively adding differentiable modules trained to match the functional gradient of the loss; moreover, the same work presented a functional gradient-based rule for reproducing kernel Hilbert spaces, which essentially corresponds to our result of Sec. 3.1. We complement this early study with modern deep learning experiments, and with our novel (weighted) softmax attention.

Furthermore, RCDL’s growing network is fundamentally a cascade of kernel machines [2, 46, 47, 26], with features in every layer trained in an end-to-end fashion through gradient descent—a core principle of deep learning. This has an interesting difference to Domingos [51]’s view on how an entire deep NN trained by gradient descent can be *approximated* by a kernel machine. In contrast, the development of RCDL builds on the *exact* layer-wise equivalence/duality between a linear layer trained by gradient descent and a kernel machine [2, 1]. This might open new avenues to study optimization in NNs.

RCDL may also open up a new path for continual learning. Unlike conventional weight-based learning which involves possibly irreversible compression during training, RCDL preserves past activations; we may potentially augment RCDL with some algorithms that revive old memories (e.g., within the forward pass of the net). Loosely, this property also aligns with the neurobiological findings on “silent memory” [52–54] where forgetting occurs due to retrieval failures (not because the actual memory content is lost) and some (optogenetic) procedure can revive the apparently lost memory.

Limitations. RCDL has several open challenges. One of the biggest concerns is the memory and compute requirements that linearly grow with the number of training datapoints, which is generally prohibitive. While some of the key-value pruning methods from the transformer literature (e.g., sliding window or token merging [55]) may provide some insights, deriving a more principled solution for RCDL remains a future research topic. At the same time, the internet (and the corresponding data storage) itself continues to grow, and memory continues to be cheaper as hardware advances—ever growing RCDL may not be completely inconceivable in a distant future. Another notable limitation is the class of base optimization algorithms currently supported by our theory; while we covered some of the basic concepts such as weight decay and momentum to augment vanilla gradient descent, we do not know yet how to derive more advanced optimizers such as Adam [38] and Muon [27] for RCDL.

Conclusion. Retrieval-centric deep learning (RCDL) aims to improve upon conventional deep learning by replacing the “linear attention” (LA) underlying the classic linear layer trained via gradient descent—a fundamental building block of deep learning—with more powerful attention mechanisms. While replacing LA with MesaNet/DeltaNet-inspired update rules yields efficient optimizers for conventional weight-based neural nets, replacing it with kernelized and softmax attention leads to unconventional nonparametric networks that learn by growing a key-value memory database instead of a fixed-size weight matrix. Using techniques from functional analysis and functional gradient methods, we derived gradient-based learning rules to train deep networks composed of the latter, and we showed empirically that nonparametric deep networks perform strongly on standard benchmarks. Our findings open a number of avenues for future work, such as investigating how to compress the growing nonparametric layers or integrate such layers into language models.

References

- [1] Kazuki Irie, Róbert Csordás, and Jürgen Schmidhuber. The dual form of neural networks revisited: Connecting test time predictions to training patterns via spotlights of attention. In *Proc. Int. Conf. on Machine Learning (ICML)*, Baltimore, MD, USA, July 2022.
- [2] Mark A. Aizerman, Emmanuil M. Braverman, and Lev I. Rozonoer. Theoretical foundations of potential function method in pattern recognition. *Automation and Remote Control*, 25(6): 917–936, 1964.
- [3] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are RNNs: Fast autoregressive transformers with linear attention. In *Proc. Int. Conf. on Machine Learning (ICML)*, Virtual only, July 2020.
- [4] Zhuoran Shen, Mingyuan Zhang, Haiyu Zhao, Shuai Yi, and Hongsheng Li. Efficient attention: Attention with linear complexities. *Preprint arXiv:1812.01243*, 2018.
- [5] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *Int. Conf. on Learning Representations (ICLR)*, San Diego, CA, USA, May 2015.
- [6] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proc. Advances in Neural Information Processing Systems (NIPS)*, pages 5998–6008, Long Beach, CA, USA, December 2017.
- [7] Kazuki Irie. The dual form of the perceptron (1964) meets deep learning and what that enables. *Preprint*, 2026.
- [8] Jürgen Schmidhuber. Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Computation*, 4(1):131–139, 1992.
- [9] Imanol Schlag, Kazuki Irie, and Jürgen Schmidhuber. Linear Transformers are secretly fast weight programmers. In *Proc. Int. Conf. on Machine Learning (ICML)*, Virtual only, July 2021.
- [10] Kazuki Irie, Imanol Schlag, Róbert Csordás, and Jürgen Schmidhuber. Going beyond linear transformers with recurrent fast weight programmers. In *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, Virtual only, December 2021.
- [11] Kazuki Irie, Francesco Faccio, and Jürgen Schmidhuber. Neural differential equations for learning to program neural nets through continuous learning rules. In *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, New Orleans, LA, USA, December 2022.
- [12] Kazuki Irie, Imanol Schlag, Róbert Csordás, and Jürgen Schmidhuber. A modern self-referential weight matrix that learns to modify itself. In *Proc. Int. Conf. on Machine Learning (ICML)*, pages 9660–9677, Baltimore, MA, USA, July 2022.
- [13] Songlin Yang, Bailin Wang, Yu Zhang, Yikang Shen, and Yoon Kim. Parallelizing linear transformers with the delta rule over sequence length. In *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, Vancouver, Canada, December 2024.
- [14] Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated delta networks: Improving Mamba2 with delta rule. In *Int. Conf. on Learning Representations (ICLR)*, Vancouver, Canada, April 2025.
- [15] Julien Siems, Timur Carstensen, Arber Zela, Frank Hutter, Massimiliano Pontil, and Riccardo Grazi. Deltaproduct: Improving state-tracking in linear RNNs via householder products. In *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, San Diego, CA, USA, December 2025.
- [16] Johannes von Oswald, Eyvind Niklasson, Ettore Randazzo, João Sacramento, Alexander Mordvintsev, Andrey Zhmoginov, and Max Vladymyrov. Transformers learn in-context by gradient descent. In *Proc. Int. Conf. on Machine Learning (ICML)*, Honolulu, HI, USA, July 2023.

- [17] Johannes von Oswald, Eyvind Niklasson, Maximilian Schlegel, Seijin Kobayashi, Nicolas Zucchet, Nino Scherrer, Nolan Miller, Mark Sandler, Max Vladymyrov, Razvan Pascanu, et al. Uncovering mesa-optimization algorithms in Transformers. *Preprint arXiv:2309.05858*, 2023.
- [18] Johannes von Oswald, Nino Scherrer, Seijin Kobayashi, Luca Versari, Songlin Yang, Maximilian Schlegel, Kaitlin Maile, Yanick Schimpf, Oliver Sieberling, Alexander Meulemans, et al. MesaNet: Sequence modeling by locally optimal test-time training. In *Int. Conf. on Learning Representations (ICLR)*, Rio de Janeiro, Brazil, 2026.
- [19] Yu Sun, Xinhao Li, Karan Dalal, Jiarui Xu, Arjun Vikram, Genghan Zhang, Yann Dubois, Xinlei Chen, Xiaolong Wang, Sanmi Koyejo, et al. Learning to (learn at test time): RNNs with expressive hidden states. In *Proc. Int. Conf. on Machine Learning (ICML)*, Vancouver, Canada, July 2025.
- [20] Ali Behrouz, Zeman Li, Praneeth Kacham, Majid Daliri, Yuan Deng, Peilin Zhong, Meisam Razaviyayn, and Vahab Mirrokni. Atlas: Learning to optimally memorize the context at test time. *Preprint arXiv:2505.23735*, 2025.
- [21] Ali Behrouz, Peilin Zhong, and Vahab Mirrokni. Titans: Learning to memorize at test time. In *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, San Diego, CA, USA, 2025.
- [22] Ali Behrouz, Meisam Razaviyayn, Peilin Zhong, and Vahab Mirrokni. Nested learning: The illusion of deep learning architectures. In *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, San Diego, CA, USA, December 2025.
- [23] Ke Alexander Wang, Jiaxin Shi, and Emily B Fox. Test-time regression: a unifying framework for designing sequence models with associative memory. *Preprint arXiv:2501.12352*, 2025.
- [24] Hao Peng, Nikolaos Pappas, Dani Yogatama, Roy Schwartz, Noah A Smith, and Lingpeng Kong. Random feature attention. In *Int. Conf. on Learning Representations (ICLR)*, Virtual only, 2021.
- [25] Songlin Yang, Bailin Wang, Yikang Shen, Rameswar Panda, and Yoon Kim. Gated linear attention transformers with hardware-efficient training. In *Proc. Int. Conf. on Machine Learning (ICML)*, Vienna, Austria, July 2024.
- [26] Bernhard Schölkopf and Alexander J Smola. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2002.
- [27] Keller Jordan, Yuchen Jin, Vlado Boza, Jiacheng You, Franz Cesista, Laker Newhouse, and Jeremy Bernstein. Muon: An optimizer for hidden layers in neural networks, 2024. URL <https://kellerjordan.github.io/posts/muon/>.
- [28] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. In *Proc. Conf. on Empirical Methods in Natural Language Processing (EMNLP)*, Lisbon, Portugal, September 2015.
- [29] Nachman Aronszajn. Theory of reproducing kernels. *Transactions of the American mathematical society*, 68(3):337–404, 1950.
- [30] Frederik Benzing. Gradient descent on neurons and its link to approximate second-order optimization. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvári, Gang Niu, and Sivan Sabato, editors, *Proc. Int. Conf. on Machine Learning (ICML)*, 2022.
- [31] James Martens and Roger B. Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In Francis R. Bach and David M. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 2408–2417. JMLR.org, 2015. URL <http://proceedings.mlr.press/v37/martens15.html>.
- [32] Bernard Widrow and Marcian E Hoff. Adaptive switching circuits. In *Proc. IRE WESCON Convention Record*, pages 96–104, Los Angeles, CA, USA, August 1960.

- [33] Biao Zhang and Rico Sennrich. Root mean square layer normalization. In *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, Vancouver, Canada, 2019.
- [34] Kunihiko Fukushima. Visual feature extraction by a multilayered network of analog threshold elements. *IEEE Transactions on Systems Science and Cybernetics*, 5(4):322–333, 1969.
- [35] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proc. Int. Conf. on Machine Learning (ICML)*, pages 807–814, Haifa, Israel, June 2010.
- [36] Dan Hendrycks. Gaussian error linear units (GELUs). *Preprint arXiv:1606.08415*, 2016.
- [37] John S Bridle. Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. *Neurocomputing: Algorithms, architectures and applications*, pages 227–236, 1990.
- [38] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Int. Conf. on Learning Representations (ICLR)*, San Diego, CA, USA, May 2015.
- [39] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *Int. Conf. on Learning Representations (ICLR)*, New Orleans, LA, USA, May 2019.
- [40] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. Neural networks for machine learning (lecture 6e). Coursera, video lectures, 2012.
- [41] Elizabeth Gardner and Bernard Derrida. Three unfinished works on the optimal storage capacity of networks. *Journal of Physics A: Mathematical and General*, 22(12), 1989.
- [42] Berfin Simsek, François Ged, Arthur Jacot, Francesco Spadaro, Clément Hongler, Wulfram Gerstner, and Johanni Brea. Geometry of the loss landscape in overparameterized neural networks: Symmetries and invariances. In *Proc. Int. Conf. on Machine Learning (ICML)*, Virtual only, July 2021.
- [43] Mor Geva, Roei Schuster, Jonathan Berant, and Omer Levy. Transformer feed-forward layers are key-value memories. In *Proc. Conf. on Empirical Methods in Natural Language Processing (EMNLP)*, pages 5484–5495, Punta Cana, Dominican Republic, November 2021.
- [44] Alex Krizhevsky. Learning multiple layers of features from tiny images. Master’s thesis, Computer Science Department, University of Toronto, 2009.
- [45] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *Int. Conf. on Learning Representations (ICLR)*, Virtual only, May 2021.
- [46] Bernhard E. Boser, Isabelle Guyon, and Vladimir Vapnik. A training algorithm for optimal margin classifiers. In *Proc. Annual ACM Conference on Computational Learning Theory (COLT)*, pages 144–152, Pittsburgh, PA, USA, July 1992. ACM.
- [47] Christopher JC Burges. A tutorial on support vector machines for pattern recognition. *Data mining and knowledge discovery*, 2(2):121–167, 1998.
- [48] Llew Mason, Jonathan Baxter, Peter L. Bartlett, and Marcus R. Frean. Boosting algorithms as gradient descent. In Sara A. Solla, Todd K. Leen, and Klaus-Robert Müller, editors, *Proc. Advances in Neural Information Processing Systems (NIPS)*, 1999.
- [49] Jerome H. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29:1189–1232, 2000.
- [50] Alexander Grubb and J. Andrew Bagnell. Boosted backpropagation learning for training deep modular networks. In Johannes Fürnkranz and Thorsten Joachims, editors, *Proc. Int. Conf. on Machine Learning (ICML)*, 2010.
- [51] Pedro Domingos. Every model learned by gradient descent is approximately a kernel machine. *Preprint arXiv:2012.00152*, 2020.

- [52] Tomás J Ryan, Dheeraj S Roy, Michele Pignatelli, Autumn Arons, and Susumu Tonegawa. Engram cells retain memory under retrograde amnesia. *Science*, 348(6238):1007–1013, 2015.
- [53] Dheeraj S Roy, Shruti Muralidhar, Lillian M Smith, and Susumu Tonegawa. Silent memory engrams as the basis for retrograde amnesia. *Proc. of National Academy of Sciences (PNAS)*, 114(46):E9972–E9979, 2017.
- [54] Samuel J Gershman, Ila Fiete, and Kazuki Irie. Key-value memory in the brain. *Neuron*, 113, 2025.
- [55] Daniel Bolya, Cheng-Yang Fu, Xiaoliang Dai, Peizhao Zhang, Christoph Feichtenhofer, and Judy Hoffman. Token merging: Your vit but faster. In *Int. Conf. on Learning Representations (ICLR)*, Kigali, Rwanda, May 2023.
- [56] Dan Hendrycks and Kevin Gimpel. Gaussian Error Linear Units (GELUs). *arXiv preprint arXiv:1606.08415*, 2016.
- [57] Jimmy Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *Preprint arXiv:1607.06450*, 2016.
- [58] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, 22(3):400–407, 1951.
- [59] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *Preprint arXiv:1410.5401*, 2014.
- [60] Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, et al. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471–476, 2016.
- [61] Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks. *Preprint arXiv:1410.3916*, 2014.
- [62] Sainbayar Sukhbaatar, Arthur Szlam, Jason Weston, and Rob Fergus. End-to-end memory networks. In *Proc. Advances in Neural Information Processing Systems (NIPS)*, pages 2440–2448, Montréal, Canada, December 2015.
- [63] Alexander Pritzel, Benigno Uria, Sriram Srinivasan, Adria Puigdomenech Badia, Oriol Vinyals, Demis Hassabis, Daan Wierstra, and Charles Blundell. Neural episodic control. In *Proc. Int. Conf. on Machine Learning (ICML)*, pages 2827–2836, Lille, France, August 2017.
- [64] Vincent-Pierre Berges, Barlas Oğuz, Daniel Haziza, Wen-tau Yih, Luke Zettlemoyer, and Gargi Ghosh. Memory layers at scale. In *Proc. Int. Conf. on Machine Learning (ICML)*, Vancouver, Canada, July 2025.
- [65] Sabri Eyuboglu, Ryan Ehrlich, Simran Arora, Neel Guha, Dylan Zinsley, Emily Liu, Will Tennien, Atri Rudra, James Zou, Azalia Mirhoseini, et al. Cartridges: Lightweight and general-purpose long context representations via self-study. *Preprint arXiv:2506.06266*, 2025.
- [66] Alan M. Turing. On computable numbers, with an application to the entscheidungsproblem. *Proc. of London Mathematical Society*, 1936.
- [67] Ali Behrouz, Zeman Li, Yuan Deng, Peilin Zhong, Meisam Razaviyayn, and Vahab Mirrokni. Memory caching: RNNs with growing memory. In *Proc. Int. Conf. on Machine Learning (ICML)*, Seoul, South Korea, July 2026.
- [68] Andrew Gordon Wilson, Zhiting Hu, Ruslan Salakhutdinov, and Eric P Xing. Deep kernel learning. In *Proc. Int. Conf. on Artificial Intelligence and Statistics (AISTATS)*, pages 370–378, Cadiz, Spain, May 2016.
- [69] Hubert Ramsauer, Bernhard Schäfl, Johannes Lehner, Philipp Seidl, Michael Widrich, Lukas Gruber, Markus Holzleitner, Thomas Adler, David Kreil, Michael K Kopp, Günter Klambauer, Johannes Brandstetter, and Sepp Hochreiter. Hopfield networks is all you need. In *Int. Conf. on Learning Representations (ICLR)*, Virtual only, May 2021.

- [70] Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, Montréal, Canada, December 2018.
- [71] Jaehoon Lee, Lechao Xiao, Samuel Schoenholz, Yasaman Bahri, Roman Novak, Jascha Sohl-Dickstein, and Jeffrey Pennington. Wide neural networks of any depth evolve as linear models under gradient descent. In *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, Vancouver, Canada, December 2019.
- [72] Lenaïc Chizat, Edouard Oyallon, and Francis Bach. On lazy training in differentiable programming. In *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, Vancouver, Canada, December 2019.
- [73] Scott Fahlman and Christian Lebiere. The cascade-correlation learning architecture. In *Proc. Advances in Neural Information Processing Systems (NIPS)*, volume 2, 1989.
- [74] John Platt. A resource-allocating network for function interpolation. *Neural Computation*, 3(2): 213–225, 1991.
- [75] Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *Preprint arXiv:1606.04671*, 2016.
- [76] Jaehong Yoon, Eunho Yang, Jeongtae Lee, and Sung Ju Hwang. Lifelong learning with dynamically expandable networks. In *Int. Conf. on Learning Representations (ICLR)*, Vancouver, Canada, April 2018.
- [77] Kenneth O Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.
- [78] Utku Evci, Bart van Merriënboer, Thomas Unterthiner, Fabian Pedregosa, and Max Vladymyrov. GradMax: Growing neural networks using gradient information. In *Int. Conf. on Learning Representations (ICLR)*, Virtual only, April 2022.
- [79] Eivinas Butkus, Kedar GarzÄgn Gupta, and Nikolaus Kriegeskorte. Growing a neural network in breadth, depth, and time. *Preprint arXiv:2605.25174*, 2026.
- [80] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, Virtual only, December 2020.
- [81] Sewon Min, Weijia Shi, Mike Lewis, Xilun Chen, Wen-tau Yih, Hannaneh Hajishirzi, and Luke Zettlemoyer. Nonparametric masked language modeling. In *Findings of the Association for Computational Linguistics*, pages 2097–2118, Toronto, Canada, July 2023.

A Derivation of the Growing Layer Algorithm for Positive Definite Kernels

In this appendix, we provide the formal derivation of the learning algorithms for the growing kernel networks introduced in Section 3.1. We establish the correspondence between the “growing database” update rules and stochastic gradient descent (SGD) performed on an implicit, infinite-dimensional weight matrix defined in a Reproducing Kernel Hilbert Space (RKHS).

A.1 The Growing Kernel Layer

A growing kernel layer with input dimension d_{in} and output dimension d_{out} is defined by a tuple $(\phi, \mathcal{D})$, where:

- $\phi : \mathbb{R}^{d_{\text{in}}} \times \mathbb{R}^{d_{\text{in}}} \rightarrow \mathbb{R}$ is a positive definite kernel.
- $\mathcal{D} = \{(\mathbf{k}_i, \mathbf{v}_i)\}_{i=1}^M$ is a finite database of key-value pairs, where $\mathbf{k}_i \in \mathbb{R}^{d_{\text{in}}}$ and $\mathbf{v}_i \in \mathbb{R}^{d_{\text{out}}}$.

The layer implements a function $f_{\phi, \mathcal{D}} : \mathbb{R}^{d_{\text{in}}} \rightarrow \mathbb{R}^{d_{\text{out}}}$ defined as:

$$f_{\phi, \mathcal{D}}(\mathbf{q}) = \sum_{(\mathbf{k}, \mathbf{v}) \in \mathcal{D}} \phi(\mathbf{k}, \mathbf{q}) \mathbf{v}. \quad (14)$$

Unlike standard parametric layers with fixed dimensions, this layer “grows” by appending new key-value pairs to $\mathcal{D}$. Deep growing kernel networks are constructed by composing such layers, interleaving them with element-wise non-linearities σ . For example, a two-layer network is represented as $F(\mathbf{x}) = \sigma(f_{\phi_2, \mathcal{D}_2}(\sigma(f_{\phi_1, \mathcal{D}_1}(\mathbf{x}))))$.

A.2 The Implicit Weight Matrix View

By the Moore-Aronszajn theorem, the positive definite kernel ϕ is associated with a Reproducing Kernel Hilbert Space (RKHS) $\mathcal{H}_\phi$ and a feature map $\psi_\phi : \mathbb{R}^{d_{\text{in}}} \rightarrow \mathcal{H}_\phi$ such that $\phi(\mathbf{x}, \mathbf{x}') = \langle \psi_\phi(\mathbf{x}), \psi_\phi(\mathbf{x}') \rangle_{\mathcal{H}_\phi}$. Since $\mathcal{H}_\phi$ is typically separable (e.g., for the RBF kernel), it is isomorphic to $\ell_2(\mathbb{N})$. We can thus view $\psi_\phi(\mathbf{x})$ as an infinite-dimensional vector.

Substituting this inner product structure into Eq. 14, we obtain:

$$\begin{aligned} f_{\phi, \mathcal{D}}(\mathbf{q}) &= \sum_{(\mathbf{k}, \mathbf{v}) \in \mathcal{D}} \mathbf{v} \langle \psi_\phi(\mathbf{k}), \psi_\phi(\mathbf{q}) \rangle_{\mathcal{H}_\phi} \\ &= \left(\sum_{(\mathbf{k}, \mathbf{v}) \in \mathcal{D}} \mathbf{v} \psi_\phi(\mathbf{k})^\top \right) \psi_\phi(\mathbf{q}) \\ &= \mathbf{W}_{\phi, \mathcal{D}} \psi_\phi(\mathbf{q}), \end{aligned} \quad (15)$$

where $\mathbf{W}_{\phi, \mathcal{D}} = \sum_{(\mathbf{k}, \mathbf{v}) \in \mathcal{D}} \mathbf{v} \psi_\phi(\mathbf{k})^\top$ is an implicit linear operator (or infinite matrix) mapping from $\mathcal{H}_\phi$ to $\mathbb{R}^{d_{\text{out}}}$. This formulation reveals that a growing kernel layer is theoretically equivalent to a linear layer with a weight matrix $\mathbf{W} \in \mathbb{R}^{d_{\text{out}} \times \infty}$ acting on the feature space, where the weight matrix has a finite rank determined by the size of $\mathcal{D}$.

A.3 Derivation of the Learning Algorithm

We now derive the update rule for the database $\mathcal{D}$ by minimizing a global loss function $\mathcal{L}$ via Stochastic Gradient Descent (SGD) on the implicit weights $\mathbf{W}$.

Consider a deep kernel network F and a specific layer ℓ with implicit weight matrix $\mathbf{W}^{(\ell)}$, kernel $\phi^{(\ell)}$, and feature map $\psi^{(\ell)}$. Let $h_{\text{in}}^{(\ell)}(\mathbf{x})$ denote the input to this layer and $h_{\text{out}}^{(\ell)}(\mathbf{x}) = \mathbf{W}^{(\ell)} \psi^{(\ell)}(h_{\text{in}}^{(\ell)}(\mathbf{x}))$ denote the output. Given a mini-batch of training data $B = \{(\mathbf{x}_b, \mathbf{y}_b)\}_{b=1}^{|B|}$, the loss is $\mathcal{J} = \sum_{b=1}^{|B|} \mathcal{L}(\mathbf{x}_b, \mathbf{y}_b; \mathbf{W}^{(\ell)})$.

The gradient of the loss with respect to the implicit matrix $\mathbf{W}^{(\ell)}$ is computed via the chain rule. Let $\mathbf{e}_b^{(\ell)}$ be the error signal (gradient of the loss with respect to the layer output) for sample b :

$$\mathbf{e}_b^{(\ell)} := \nabla_{h_{\text{out}}^{(\ell)}} \mathcal{L}(\mathbf{x}_b, \mathbf{y}_b) \in \mathbb{R}^{d_{\text{out}}^{(\ell)}}. \quad (16)$$

The gradient with respect to $\mathbf{W}^{(\ell)}$ is the outer product of the error signal and the layer input features:

$$\nabla_{\mathbf{W}^{(\ell)}} \mathcal{J} = \sum_{b=1}^{|B|} \mathbf{e}_b^{(\ell)} \psi^{(\ell)}(h_{\text{in}}^{(\ell)}(\mathbf{x}_b))^\top. \quad (17)$$

Applying the SGD update rule $\mathbf{W}_{t+1}^{(\ell)} = \mathbf{W}_t^{(\ell)} - \eta \nabla_{\mathbf{W}^{(\ell)}} \mathcal{J}$ yields:

$$\mathbf{W}_{t+1}^{(\ell)} = \mathbf{W}_t^{(\ell)} - \eta \sum_{b=1}^{|B|} \mathbf{e}_b^{(\ell)} \psi^{(\ell)}(h_{\text{in}}^{(\ell)}(\mathbf{x}_b))^\top. \quad (18)$$

We now examine the function computed by the layer at step $t + 1$ for an arbitrary query $\mathbf{q}$:

$$\begin{aligned} f_{t+1}^{(\ell)}(\mathbf{q}) &= \mathbf{W}_{t+1}^{(\ell)} \psi^{(\ell)}(\mathbf{q}) \\ &= \mathbf{W}_t^{(\ell)} \psi^{(\ell)}(\mathbf{q}) + \sum_{b=1}^{|B|} \left(-\eta \mathbf{e}_b^{(\ell)} \right) \psi^{(\ell)}(h_{\text{in}}^{(\ell)}(\mathbf{x}_b))^\top \psi^{(\ell)}(\mathbf{q}) \\ &= f_t^{(\ell)}(\mathbf{q}) + \sum_{b=1}^{|B|} \mathbf{v}_{\text{new},b} \phi^{(\ell)}(\mathbf{k}_{\text{new},b}, \mathbf{q}), \end{aligned} \quad (19)$$

where $\mathbf{k}_{\text{new},b} = h_{\text{in}}^{(\ell)}(\mathbf{x}_b)$ and $\mathbf{v}_{\text{new},b} = -\eta \mathbf{e}_b^{(\ell)}$. This confirms that performing SGD on the implicit infinite-dimensional weight matrix is exactly equivalent to appending the current batch inputs (as keys) and their scaled negative gradients (as values) to the database $\mathcal{D}$.

A.4 Optimization Dynamics: Weight Decay and Momentum

We extend the framework to include weight decay and momentum, deriving their dual forms in the growing database context.

A.4.1 Weight Decay and Database Pruning

Consider the L_2 -regularized objective $\mathcal{L}_{\text{reg}}(\mathbf{W}) = \mathcal{L}(\mathbf{W}) + \frac{\gamma}{2\eta} \|\mathbf{W}\|_{\text{HS}}^2$, where $\|\cdot\|_{\text{HS}}$ is the Hilbert-Schmidt (Frobenius) norm and $\gamma \in (0, 1)$ is the decay factor. The SGD update becomes:

$$\mathbf{W}_{t+1} = (1 - \gamma) \mathbf{W}_t - \eta \nabla_{\mathbf{W}} \mathcal{L}. \quad (20)$$

In the functional form $f(\mathbf{q}) = \mathbf{W} \psi(\mathbf{q})$, the scaling factor $(1 - \gamma)$ distributes linearly over the sum of key-value pairs. Thus, weight decay is implemented by scaling down the value vectors of *all* existing pairs in the database at each step:

$$\mathcal{D}_{t+1} = \{(\mathbf{k}, (1 - \gamma)\mathbf{v}) \mid (\mathbf{k}, \mathbf{v}) \in \mathcal{D}_t\} \cup \{(\mathbf{k}_{\text{new}}, \mathbf{v}_{\text{new}})\}. \quad (21)$$

Pruning: This update implies that the magnitude of a value vector stored at time t_0 decays as $\|\mathbf{v}(t)\| = (1 - \gamma)^{t-t_0} \|\mathbf{v}(t_0)\|$. This justifies a principled pruning strategy: once $\|\mathbf{v}(t)\|$ drops below a threshold ϵ , the pair's contribution becomes negligible, and it can be removed from memory, bounding the database size.

A.4.2 Momentum

We adopt the standard momentum formulation where a velocity matrix $\mathbf{V}_t$ accumulates gradients. Let $\mathbf{G}_t = -\eta \nabla_{\mathbf{W}} \mathcal{L}_t$ be the update term at step t . The velocity update is $\mathbf{V}_{t+1} = \mu \mathbf{V}_t + \mathbf{G}_t$, and the weight update is $\mathbf{W}_{t+1} = \mathbf{W}_t + \mathbf{V}_{t+1}$. Unrolling the recurrence, the weight matrix at time T is a linear combination of all past update terms $\mathbf{G}_\tau$ (created at time $\tau < T$):

$$\mathbf{W}_T = \sum_{\tau=1}^T c(T, \tau) \mathbf{G}_\tau, \quad \text{with } c(T, \tau) = \sum_{k=0}^{T-\tau} \mu^k = \frac{1 - \mu^{T-\tau+1}}{1 - \mu}. \quad (22)$$

Since each $\mathbf{G}_\tau$ corresponds to the batch of key-value pairs created at time τ , we do not need to store explicit velocity vectors. Instead, we store the creation timestamp t_{create} with each pair. During the forward pass at time T , we dynamically compute the coefficient $c(T, t_{\text{create}})$:

$$f_T(\mathbf{q}) = \sum_{(\mathbf{k}, \mathbf{v}_{\text{init}}, t_{\text{create}}) \in \mathcal{D}} \left(\frac{1 - \mu^{T-t_{\text{create}}+1}}{1 - \mu} \right) \phi(\mathbf{k}, \mathbf{q}) \mathbf{v}_{\text{init}}. \quad (23)$$

This allows exact momentum computation with no additional memory overhead per pair. Combining both techniques results in decaying the stored values for regularization while scaling the retrieval output for momentum.

B Further Details and Remarks for the Softmax Case

B.1 Remark: the functional space of weighted softmax functions

In order to make the softmax function in Eq. 3 well-defined, we require that

$$\int_{\mathbb{R}^{d_{\text{in}}} \times \mathbb{R}^{d_{\text{out}}}} w(\mathbf{k}, \mathbf{v}) e^{\langle \mathbf{k}, \mathbf{q} \rangle + \langle \mathbf{v}, \mathbf{v}' \rangle} d\mathbf{k} d\mathbf{v} < \infty, \\ \forall \mathbf{q} \in \mathbb{R}^{d_{\text{in}}}, \forall \mathbf{v}' \in \mathbb{R}^{d_{\text{out}}}.$$

In other words, if we interpret w as the density of a measure μ_w on $\mathbb{R}^{d_{\text{in}}} \times \mathbb{R}^{d_{\text{out}}}$, then we require that the moment-generating function of μ_w be well-defined and finite everywhere.

Remark B.1. In the case of a finite database of weighted key-value pairs, if we have several tuples having the same key, then we can combine them into one tuple. For example, we can combine $(w_1, \mathbf{k}, \mathbf{v}_1)$ and $(w_2, \mathbf{k}, \mathbf{v}_2)$ and replace them with $(w_1 + w_2, \mathbf{k}, \frac{w_1 \mathbf{v}_1 + w_2 \mathbf{v}_2}{w_1 + w_2})$, and this will induce the same softmax function. Analogously, if we define the “marginalized” key-weight function $\tilde{w} : \mathbb{R}^{d_{\text{in}}} \rightarrow \mathbb{R}^+$ as

$$\tilde{w}(\mathbf{k}) = \int_{\mathbb{R}^{d_{\text{out}}}} w(\mathbf{k}, \mathbf{v}) d\mathbf{v},$$

and the marginalized value function $\tilde{v} : \mathbb{R}^{d_{\text{in}}} \rightarrow \mathbb{R}^{d_{\text{out}}}$ which associates a unique value for every key as

$$\tilde{v}(\mathbf{k}) := \frac{1}{\tilde{w}(\mathbf{k})} \int_{\mathbb{R}^{d_{\text{out}}}} w(\mathbf{k}, \mathbf{v}) \mathbf{v} d\mathbf{v},$$

then we can write s_w in terms of $\tilde{w}$ and $\tilde{v}$ as follows:

$$s_w(\mathbf{q}) = s_{\tilde{w}, \tilde{v}}(\mathbf{q}) := \frac{\int_{\mathbb{R}^{d_{\text{in}}}} \tilde{w}(\mathbf{k}) e^{\langle \mathbf{k}, \mathbf{q} \rangle} \tilde{v}(\mathbf{k}) d\mathbf{k}}{\int_{\mathbb{R}^{d_{\text{in}}}} \tilde{w}(\mathbf{k}) e^{\langle \mathbf{k}, \mathbf{q} \rangle} d\mathbf{k}}. \quad (24)$$

B.2 Continuous softmax function induced by the Gaussian mixture density & Discrete weighted softmax function

Here we provide a full derivation demonstrating that the continuous softmax function induced by the Gaussian mixture density $w_{\mathcal{D}}$ is exactly equivalent to the discrete weighted softmax function $s_{\mathcal{D}}$ defined by the database $\mathcal{D}$.

$$\begin{aligned} s_{w_{\mathcal{D}}}(\mathbf{q}) &= \frac{\int_{\mathbb{R}^{d_{\text{in}}} \times \mathbb{R}^{d_{\text{out}}}} w_{\mathcal{D}}(\mathbf{k}, \mathbf{v}) e^{\langle \mathbf{k}, \mathbf{q} \rangle} \mathbf{v} d\mathbf{k} d\mathbf{v}}{\int_{\mathbb{R}^{d_{\text{in}}} \times \mathbb{R}^{d_{\text{out}}}} w_{\mathcal{D}}(\mathbf{k}, \mathbf{v}) e^{\langle \mathbf{k}, \mathbf{q} \rangle} d\mathbf{k} d\mathbf{v}} \\ &= \frac{\sum_{i=1}^n w_i \mathbb{E}_{(\mathbf{k}, \mathbf{v}) \sim \mathcal{N}((\mathbf{k}_i, \mathbf{v}_i), \Sigma)} [e^{\langle \mathbf{k}, \mathbf{q} \rangle} \mathbf{v}]}{\sum_{i=1}^n w_i \mathbb{E}_{(\mathbf{k}, \mathbf{v}) \sim \mathcal{N}((\mathbf{k}_i, \mathbf{v}_i), \Sigma)} [e^{\langle \mathbf{k}, \mathbf{q} \rangle}]} \\ &= \frac{\sum_{i=1}^n w_i \mathbb{E}_{\mathbf{k} \sim \mathcal{N}(\mathbf{k}_i, \sigma_k^2 \mathbf{I})} [e^{\langle \mathbf{k}, \mathbf{q} \rangle}] \mathbb{E}_{\mathbf{v} \sim \mathcal{N}(\mathbf{v}_i, \sigma_v^2 \mathbf{I})} [\mathbf{v}]}{\sum_{i=1}^n w_i \mathbb{E}_{\mathbf{k} \sim \mathcal{N}(\mathbf{k}_i, \sigma_k^2 \mathbf{I})} [e^{\langle \mathbf{k}, \mathbf{q} \rangle}]} \\ &= \frac{\sum_{i=1}^n w_i e^{\langle \mathbf{k}_i, \mathbf{q} \rangle + \frac{1}{2} \sigma_k^2 \|\mathbf{q}\|^2} \mathbf{v}_i}{\sum_{i=1}^n w_i e^{\langle \mathbf{k}_i, \mathbf{q} \rangle + \frac{1}{2} \sigma_k^2 \|\mathbf{q}\|^2}} \\ &= \frac{e^{\frac{1}{2} \sigma_k^2 \|\mathbf{q}\|^2} \sum_{i=1}^n w_i e^{\langle \mathbf{k}_i, \mathbf{q} \rangle} \mathbf{v}_i}{e^{\frac{1}{2} \sigma_k^2 \|\mathbf{q}\|^2} \sum_{i=1}^n w_i e^{\langle \mathbf{k}_i, \mathbf{q} \rangle}} \\ &= \frac{\sum_{i=1}^n w_i e^{\langle \mathbf{k}_i, \mathbf{q} \rangle} \mathbf{v}_i}{\sum_{i=1}^n w_i e^{\langle \mathbf{k}_i, \mathbf{q} \rangle}} = s_{\mathcal{D}}(\mathbf{q}). \end{aligned}$$

B.3 Further discussion on the standard (unweighted) softmax case

In Sec. 3.2 of the main text, we introduced the weighted softmax by noting that deriving a principled learning rule for the standard/unweighted softmax is non-trivial due to the discontinuous nature. Here we illustrate this difficulty by presenting an example of a naive derivation of a learning rule for the standard softmax case and showing where it breaks down.

Model description. Consider a growing softmax layer that, at every training step t , maintains a key-value memory state consisting of two matrices $\mathbf{K}_{t-1} = [\mathbf{k}_1, \dots, \mathbf{k}_{t-1}] \in \mathbb{R}^{d_{\text{in}} \times (t-1)}$ and $\mathbf{V}_{t-1} = [\mathbf{v}_1, \dots, \mathbf{v}_{t-1}] \in \mathbb{R}^{d_{\text{out}} \times (t-1)}$ (both to be defined later), receives a layer input $\mathbf{x}_t \in \mathbb{R}^{d_{\text{in}}}$ and produces a layer output $\mathbf{y}_t \in \mathbb{R}^{d_{\text{out}}}$ through the softmax attention function:

$$\mathbf{q}_t = \mathbf{x}_t \quad (25)$$

$$\mathbf{y}_t = \mathbf{V}_{t-1} \text{softmax}(\mathbf{K}_{t-1}^\top \mathbf{q}_t) = \sum_{\tau=1}^{t-1} \alpha_{t-1,\tau} \mathbf{v}_\tau \quad (26)$$

where $\alpha_{t-1,\tau} \in \mathbb{R}$ denotes the attention score for the key $\mathbf{k}_\tau$ using the query $\mathbf{q}_t$ and the system's key-value memory state from step $t-1$ (i.e., before updating the system at step t). That is, $\mathbf{x}_t$ is used as the query $\mathbf{q}_t$ in the attention operation to retrieve from the key-value memory consisting of matrices $\mathbf{K}_{t-1}$ and $\mathbf{V}_{t-1}$.

Then, the system's memory state is updated as:

$$\mathbf{K}_t = [\mathbf{K}_{t-1}, \mathbf{k}_t] \in \mathbb{R}^{d_{\text{in}} \times t} \quad (27)$$

$$\mathbf{V}_t = [\mathbf{V}_{t-1}, \mathbf{v}_t] \in \mathbb{R}^{d_{\text{out}} \times t} \quad (28)$$

where $[]$ denotes concatenation along the time dimension.

By following the linear case (Sec. 2), let us set the key to be the layer input, i.e., $\mathbf{k}_t = \mathbf{x}_t$. Now the question is how should we set $\mathbf{v}_t \in \mathbb{R}^{d_{\text{out}}}$ so that this memory update corresponds to a principled learning algorithm for this system in a way that it optimizes a given objective function $\mathcal{L}$?

Direct Derivation. For that, we first set the new value vector $\mathbf{v}_t \in \mathbb{R}^{d_{\text{out}}}$ as an arbitrary free parameter vector, and consider the virtually updated system with the newly added key-value pair $(\mathbf{x}_t, \mathbf{v}_t)$. After this update, the output of the system on the input $\mathbf{q}_t = \mathbf{x}_t$ becomes:

$$\mathbf{y}_t^{\text{new}} = \mathbf{V}_t \text{softmax}(\mathbf{K}_t^\top \mathbf{q}_t) = \sum_{\tau=1}^{t-1} \alpha_{t,\tau}^{\text{new}} \mathbf{v}_\tau + \alpha_{t,t}^{\text{new}} \mathbf{v}_t \quad (29)$$

that is, compared to the old system of Eq. 26, one extra term (the last term in Eq. 29) is introduced in the sum as we now have one more value vector, with newly computed attention scores $\alpha_{t,\tau}^{\text{new}}$ normalized over the new set of keys, including the attention score $\alpha_{t,t}^{\text{new}}$ for the newly added key $\mathbf{x}_t$.

Our goal is to improve the loss on input $\mathbf{q}_t$ by adding the new value token $\mathbf{v}_t$ to the system. For that, we can apply a step of gradient descent on loss $\mathcal{L}$ w.r.t. $\mathbf{v}_t$ with a learning rate $\eta_t \in \mathbb{R}_{>0}$, which yields:

$$\mathbf{v}_t = \mathbf{v}^{\text{old}} - \eta_t \frac{\partial \mathcal{L}_t(\mathbf{y}_t^{\text{new}})}{\partial \mathbf{v}} \quad (30)$$

where two more variables, $\mathbf{v}^{\text{old}}$ and the gradient term, are still to be determined; as we see below, this raises difficulties we cannot trivially overcome.

First, what should $\mathbf{v}^{\text{old}}$ be? Essentially, $\mathbf{v}^{\text{old}}$ is the value for newly added $\mathbf{v}_t$ that preserves the old system (representing the initial system before applying a step of gradient descent). For this, we can express the output of the new system $\mathbf{y}_t^{\text{new}}$ (Eq. 29) as a function of that of the old one $\mathbf{y}_t$ (Eq. 26) and search for the value of $\mathbf{v}_t$ that makes them equal. The apparently tricky difference between them is the modified attention scores: $\alpha_{t,\tau}^{\text{new}}$ (Eq. 29) vs. $\alpha_{t-1,\tau}$ (Eq. 26) for $1 \leq \tau \leq t-1$. However, thankfully, we can notice a simple relation that connects the old attention scores before the update ($\alpha_{t-1,\tau}$) and the new ones ($\alpha_{t,\tau}^{\text{new}}$) after the update. By denoting Z^{old} and Z^{new} the denominators (normalizers) of the attention scores when the input is $\mathbf{q}_t$, before and after the update, respectively, i.e.,

$$Z^{\text{old}} = \sum_{\tau=1}^{t-1} e^{\mathbf{x}_\tau^\top \mathbf{q}_t}; \quad Z^{\text{new}} = \sum_{\tau=1}^t e^{\mathbf{x}_\tau^\top \mathbf{q}_t} = Z^{\text{old}} + e^{\mathbf{q}_t^\top \mathbf{q}_t}, \quad (31)$$

we can express the new attention scores as a function of the old one:

$$\alpha_{t,\tau}^{\text{new}} = \frac{e^{\mathbf{x}_\tau^\top \mathbf{q}_t}}{Z^{\text{new}}} = \frac{Z^{\text{old}}}{Z^{\text{new}}} \times \frac{e^{\mathbf{x}_\tau^\top \mathbf{q}_t}}{Z^{\text{old}}} = \frac{Z^{\text{old}}}{Z^{\text{new}}} \alpha_{t-1,\tau}, \quad (32)$$

and additionally, the normalizer ratio can also be simplified, since $Z^{\text{new}} = Z^{\text{old}} + e^{\mathbf{q}_t^\top \mathbf{q}_t}$, as follows:

$$\frac{Z^{\text{old}}}{Z^{\text{new}}} = \frac{Z^{\text{old}} + e^{\mathbf{q}_t^\top \mathbf{q}_t} - e^{\mathbf{q}_t^\top \mathbf{q}_t}}{Z^{\text{new}}} = \frac{Z^{\text{new}} - e^{\mathbf{q}_t^\top \mathbf{q}_t}}{Z^{\text{new}}} = 1 - \frac{e^{\mathbf{q}_t^\top \mathbf{q}_t}}{Z^{\text{new}}} = 1 - \alpha_{t,t}^{\text{new}}. \quad (33)$$

By inserting this ratio to Eq. 32, it yields:

$$\alpha_{t,\tau}^{\text{new}} = (1 - \alpha_{t,t}^{\text{new}}) \alpha_{t-1,\tau}, \quad (34)$$

that is, the new attention scores for the old keys shrink by factor $(1 - \alpha_{t,t}^{\text{new}})$. This allows us to express the new output $\mathbf{y}_t^{\text{new}}$ as a function of the old one $\mathbf{y}_t$:

$$\mathbf{y}_t^{\text{new}} = \sum_{\tau=1}^{t-1} \alpha_{t,\tau}^{\text{new}} \mathbf{v}_\tau + \alpha_{t,t}^{\text{new}} \mathbf{v} \quad (29)$$

$$= \sum_{\tau=1}^{t-1} (1 - \alpha_{t,t}^{\text{new}}) \alpha_{t-1,\tau} \mathbf{v}_\tau + \alpha_{t,t}^{\text{new}} \mathbf{v} = (1 - \alpha_{t,t}^{\text{new}}) \left(\sum_{\tau=1}^{t-1} \alpha_{t-1,\tau} \mathbf{v}_\tau \right) + \alpha_{t,t}^{\text{new}} \mathbf{v} \quad (35)$$

$$= (1 - \alpha_{t,t}^{\text{new}}) \mathbf{y}_t + \alpha_{t,t}^{\text{new}} \mathbf{v} \quad (36)$$

$$= \mathbf{y}_t + \alpha_{t,t}^{\text{new}} (\mathbf{v} - \mathbf{y}_t). \quad (37)$$

Here we can at least find $\mathbf{v}_t = \mathbf{v}^{\text{old}}$ such that the updated system reproduces the old system's output on input $\mathbf{q}_t$, that is, $\mathbf{y}_t^{\text{new}} = \mathbf{y}_t$, which, from Eq. 37 holds when $\mathbf{v}_t = \mathbf{v}^{\text{old}} = \mathbf{y}_t$, canceling the last term. However, this only preserves the mapping for the current query $\mathbf{q}_t$; as the new key $\mathbf{k}_t$ alters the softmax denominator globally, the output for any other query will be modified, failing to achieve the global function preservation.

Second, while the computation of the gradient term seems straightforward:

$$\frac{\partial \mathcal{L}_t(\mathbf{y}_t^{\text{new}})}{\partial \mathbf{v}} = \frac{\partial \mathcal{L}_t(\mathbf{y}_t^{\text{new}})}{\partial \mathbf{y}_t^{\text{new}}} \times \frac{\partial \mathbf{y}_t^{\text{new}}}{\partial \mathbf{v}} = (\nabla_{\mathbf{y}_t^{\text{new}}} \mathcal{L}_t) \alpha_{t,t}^{\text{new}}, \quad (38)$$

by differentiating Eq. 29 w.r.t. $\mathbf{v}$, this requires access to the *post-update* gradient of the loss w.r.t. the *new* output $(\nabla_{\mathbf{y}_t^{\text{new}}} \mathcal{L}_t)$. While this creates a circular dependency in the general case, by setting $\mathbf{v}_t = \mathbf{y}_t$ as prescribed by the derivation above, $\nabla_{\mathbf{y}_t^{\text{new}}} \mathcal{L}_t$ can be essentially replaced by $\nabla_{\mathbf{y}_t} \mathcal{L}_t$.

Putting these together, we obtain the following update rule:

$$\mathbf{v}_t = \mathbf{v}^{\text{old}} - \eta_t \frac{\partial \mathcal{L}_t(\mathbf{y}_t^{\text{new}})}{\partial \mathbf{v}} = \mathbf{y}_t - \eta_t \alpha_{t,t}^{\text{new}} \nabla_{\mathbf{y}_t^{\text{new}}} \mathcal{L}_t, \quad (39)$$

where $\alpha_{t,t}^{\text{new}} = \frac{e^{\mathbf{x}_t^\top \mathbf{x}_t}}{Z^{\text{new}}} \in \mathbb{R}$ can be computed simply as we have $Z^{\text{new}} = Z^{\text{old}} + e^{\mathbf{x}_t^\top \mathbf{x}_t}$ for the normalizer, and Z^{old} is computed to compute $\mathbf{y}_t$.

However, this is not a satisfactory derivation as the choice of $\mathbf{v}^{\text{old}}$ is limited to the local function preservation on one data point. This derivation illustrates that a naive approach that consists in parameterizing the value vector as a free parameter and applying gradient descent on it does not work out of the box.

C Functional Gradient Descent in the Weighted Softmax Space

In this appendix, we rigorously derive the functional gradient of the loss with respect to the weight function $w \in \mathcal{H}_{\sigma_k, \sigma_v}$ and show that the gradient descent update corresponds to adding a new key-value tuple to the database.

C.1 Setup and Definitions

Let $\mathcal{H} = \mathcal{H}_{\sigma_k, \sigma_v}$ be the Hilbert space of weight functions $w : \mathbb{R}^{d_{\text{in}}} \times \mathbb{R}^{d_{\text{out}}} \rightarrow \mathbb{R}$ equipped with the inner product $\langle f, g \rangle_{\mathcal{H}} = \int f(\mathbf{k}, \mathbf{v})g(\mathbf{k}, \mathbf{v})d\mu(\mathbf{k}, \mathbf{v})$ and norm $\|w\|_{\mathcal{H}}$, where $\mu = \mathcal{N}(\mathbf{0}, \Sigma)$ is the centered Gaussian measure with block covariance $\Sigma = \text{diag}(\Sigma_k, \Sigma_v) = \text{diag}(\sigma_k^2 \mathbf{I}, \sigma_v^2 \mathbf{I})$. Recall the definition of the softmax functional

$$s_w(\mathbf{q}) = \frac{\int_{\mathbb{R}^{d_{\text{in}}} \times \mathbb{R}^{d_{\text{out}}}} w(\mathbf{k}, \mathbf{v}) e^{\langle \mathbf{k}, \mathbf{q} \rangle} \mathbf{v} d\mu(\mathbf{k}, \mathbf{v})}{\int_{\mathbb{R}^{d_{\text{in}}} \times \mathbb{R}^{d_{\text{out}}}} w(\mathbf{k}, \mathbf{v}) e^{\langle \mathbf{k}, \mathbf{q} \rangle} d\mu(\mathbf{k}, \mathbf{v})} = \frac{N(w)}{D(w)},$$

where we define the denominator functional $D : \mathcal{H} \rightarrow \mathbb{R}$ and the numerator operator $N : \mathcal{H} \rightarrow \mathbb{R}^{d_{\text{out}}}$ as follows:

$$D(w) = \int_{\mathbb{R}^{d_{\text{in}}} \times \mathbb{R}^{d_{\text{out}}}} w(\mathbf{k}, \mathbf{v}) e^{\langle \mathbf{k}, \mathbf{q} \rangle} d\mu(\mathbf{k}, \mathbf{v}), \quad (40)$$

$$N(w) = \int_{\mathbb{R}^{d_{\text{in}}} \times \mathbb{R}^{d_{\text{out}}}} w(\mathbf{k}, \mathbf{v}) e^{\langle \mathbf{k}, \mathbf{q} \rangle} \mathbf{v} d\mu(\mathbf{k}, \mathbf{v}). \quad (41)$$

Let $N_j(w)$ denote the j -th component of the vector $N(w)$. It is a scalar linear functional:

$$N_j(w) = \int w(\mathbf{k}, \mathbf{v}) e^{\langle \mathbf{k}, \mathbf{q} \rangle} v_j d\mu(\mathbf{k}, \mathbf{v}).$$

By defining the kernels

$$K_D(\mathbf{k}, \mathbf{v}) = e^{\langle \mathbf{k}, \mathbf{q} \rangle}, \quad K_{N,j}(\mathbf{k}, \mathbf{v}) = e^{\langle \mathbf{k}, \mathbf{q} \rangle} v_j,$$

we can rewrite D and N_j as

$$D(w) = \int_{\mathbb{R}^{d_{\text{in}}} \times \mathbb{R}^{d_{\text{out}}}} w(\mathbf{k}, \mathbf{v}) K_D(\mathbf{k}, \mathbf{v}) d\mu(\mathbf{k}, \mathbf{v}), \quad (42)$$

$$N_j(w) = \int_{\mathbb{R}^{d_{\text{in}}} \times \mathbb{R}^{d_{\text{out}}}} w(\mathbf{k}, \mathbf{v}) K_{N,j}(\mathbf{k}, \mathbf{v}) d\mu(\mathbf{k}, \mathbf{v}). \quad (43)$$

By the Riesz Representation Theorem, if the functions representing these kernels belong to $\mathcal{H}$, the functionals are bounded.

We verify membership in $\mathcal{H}$ by checking the condition $\mathbb{E}_{\mu}[f(\mathbf{k}, \mathbf{v})^2 e^{\langle \mathbf{k}, \mathbf{u} \rangle + \langle \mathbf{v}, \mathbf{z} \rangle}] < \infty$ for all $\mathbf{u}, \mathbf{z}$.

For K_D , we have

$$\|K_D\|_{\mathcal{H}}^2 = \mathbb{E}_{(\mathbf{k}, \mathbf{v}) \sim \mu} \left[(e^{\langle \mathbf{k}, \mathbf{q} \rangle})^2 e^{\langle \mathbf{k}, \mathbf{u} \rangle + \langle \mathbf{v}, \mathbf{z} \rangle} \right] = \mathbb{E}_{\mathbf{k}}[e^{\langle \mathbf{k}, 2\mathbf{q} + \mathbf{u} \rangle}] \mathbb{E}_{\mathbf{v}}[e^{\langle \mathbf{v}, \mathbf{z} \rangle}].$$

These are moment generating functions of Gaussian variables, which are finite everywhere.

For $K_{N,j}$, we have

$$\|K_{N,j}\|_{\mathcal{H}}^2 = \mathbb{E}_{(\mathbf{k}, \mathbf{v}) \sim \mu} \left[(e^{\langle \mathbf{k}, \mathbf{q} \rangle} v_j)^2 e^{\langle \mathbf{k}, \mathbf{u} \rangle + \langle \mathbf{v}, \mathbf{z} \rangle} \right] = \mathbb{E}_{\mathbf{k}}[e^{\langle \mathbf{k}, 2\mathbf{q} + \mathbf{u} \rangle}] \mathbb{E}_{\mathbf{v}}[v_j^2 e^{\langle \mathbf{v}, \mathbf{z} \rangle}].$$

The second term involves the second moment of a Gaussian under an exponential tilt, which is also finite. Thus, $D(w)$ and $N(w)$ are bounded linear operators. Specifically, there exist constants $C_N, C_D < \infty$ such that for any $\delta \in \mathcal{H}$:

$$\|N(\delta)\|_2 \leq C_N \|\delta\|_{\mathcal{H}}, \quad |D(\delta)| \leq C_D \|\delta\|_{\mathcal{H}} \quad (44)$$

C.2 Fréchet Differentiability of s_w

We claim that the Fréchet derivative of s_w at w , denoted ds_w , is the linear operator acting on a perturbation $\delta \in \mathcal{H}$ as:

$$ds_w(\delta) = \frac{N(\delta)D(w) - N(w)D(\delta)}{D(w)^2} \quad (45)$$

To prove this, we must show that the remainder term $R(\delta) = s_{w+\delta}(\mathbf{q}) - s_w(\mathbf{q}) - ds_w(\delta)$ satisfies $\|R(\delta)\| = o(\|\delta\|_{\mathcal{H}})$ as $\|\delta\|_{\mathcal{H}} \rightarrow 0$.

Proof. Let us expand the difference $s_{w+\delta} - s_w$:

$$s_{w+\delta} - s_w = \frac{N(w) + N(\delta)}{D(w) + D(\delta)} - \frac{N(w)}{D(w)} \quad (46)$$

$$= \frac{D(w)(N(w) + N(\delta)) - N(w)(D(w) + D(\delta))}{D(w)(D(w) + D(\delta))} \quad (47)$$

$$= \frac{D(w)N(\delta) - N(w)D(\delta)}{D(w)^2 + D(w)D(\delta)} \quad (48)$$

Subtracting the candidate derivative $ds_w(\delta)$:

$$R(\delta) = \frac{D(w)N(\delta) - N(w)D(\delta)}{D(w)(D(w) + D(\delta))} - \frac{D(w)N(\delta) - N(w)D(\delta)}{D(w)^2} \quad (49)$$

$$= [D(w)N(\delta) - N(w)D(\delta)] \left(\frac{1}{D(w)(D(w) + D(\delta))} - \frac{1}{D(w)^2} \right) \quad (50)$$

$$= [D(w)N(\delta) - N(w)D(\delta)] \left(\frac{D(w) - (D(w) + D(\delta))}{D(w)^2(D(w) + D(\delta))} \right) \quad (51)$$

$$= -[D(w)N(\delta) - N(w)D(\delta)] \frac{D(\delta)}{D(w)^2(D(w) + D(\delta))} \quad (52)$$

Now we bound the norm of $R(\delta)$. Let w be fixed such that $D(w) \neq 0$. For sufficiently small δ (specifically $\|\delta\|_{\mathcal{H}} < |D(w)|/(2C_D)$), we have

$$|D(w) + D(\delta)| \geq |D(w)| - |D(\delta)| \geq |D(w)| - C_D \|\delta\|_{\mathcal{H}} \geq |D(w)|/2.$$

Furthermore,

$$\|D(w)N(\delta) - N(w)D(\delta)\| \leq |D(w)|\|N(\delta)\| + \|N(w)\|D(\delta) \quad (53)$$

$$\leq (|D(w)|C_N + \|N(w)\|C_D)\|\delta\|_{\mathcal{H}} \quad (54)$$

$$= C_1 \|\delta\|_{\mathcal{H}}. \quad (55)$$

Substituting these back into the expression for $R(\delta)$:

$$\|R(\delta)\| \leq \frac{C_1 \|\delta\|_{\mathcal{H}} \cdot |D(\delta)|}{|D(w)|^2 |D(w) + D(\delta)|} \quad (56)$$

$$\leq \frac{C_1 \|\delta\|_{\mathcal{H}} \cdot C_D \|\delta\|_{\mathcal{H}}}{|D(w)|^2 (|D(w)|/2)} \quad (57)$$

$$= \frac{2C_1 C_D}{|D(w)|^3} \|\delta\|_{\mathcal{H}}^2 \quad (58)$$

$$= O(\|\delta\|_{\mathcal{H}}^2). \quad (59)$$

Since $\|R(\delta)\| \leq C \|\delta\|_{\mathcal{H}}^2$, it implies $\lim_{\|\delta\| \rightarrow 0} \frac{\|R(\delta)\|}{\|\delta\|} = 0$. Thus, s_w is Fréchet differentiable. $\square$

C.3 Derivation of the Functional Gradient

Let $\mathcal{L}(\hat{\mathbf{y}}, \mathbf{y})$ be a loss function that is differentiable with respect to the prediction $\hat{\mathbf{y}}$. By the chain rule for Fréchet derivatives, the composition $J(w) = \mathcal{L}(s_w(\mathbf{q}), \mathbf{y})$ is differentiable with respect to w , and its differential is:

$$dJ(\delta) = \langle \nabla_{\hat{\mathbf{y}}} \mathcal{L}, ds_w(\delta) \rangle \quad (60)$$

Substituting $ds_w(\delta)$:

$$dJ(\delta) = \left\langle \nabla_{\hat{\mathbf{y}}} \mathcal{L}, \frac{N(\delta)}{D(w)} - s_w(\mathbf{q}) \frac{D(\delta)}{D(w)} \right\rangle \quad (61)$$

$$= \frac{1}{D(w)} \langle \nabla_{\hat{\mathbf{y}}} \mathcal{L}, N(\delta) \rangle - \frac{\langle \nabla_{\hat{\mathbf{y}}} \mathcal{L}, s_w(\mathbf{q}) \rangle}{D(w)} D(\delta). \quad (62)$$

Using the integral definitions of $N(\delta)$ and $D(\delta)$ and the linearity of the inner product, we can rewrite this as an inner product in $\mathcal{H}$:

$$dJ(\delta) = \int \delta(\mathbf{k}, \mathbf{v}) \left[\frac{1}{D(w)} \langle \nabla_{\hat{\mathbf{y}}} \mathcal{L}, \mathbf{v} \rangle e^{\langle \mathbf{k}, \mathbf{q} \rangle} - \frac{\langle \nabla_{\hat{\mathbf{y}}} \mathcal{L}, s_w(\mathbf{q}) \rangle}{D(w)} e^{\langle \mathbf{k}, \mathbf{q} \rangle} \right] d\mu \quad (63)$$

$$= \int \delta(\mathbf{k}, \mathbf{v}) \frac{e^{\langle \mathbf{k}, \mathbf{q} \rangle}}{D(w)} \langle \nabla_{\hat{\mathbf{y}}} \mathcal{L}, \mathbf{v} - s_w(\mathbf{q}) \rangle d\mu \quad (64)$$

$$= \langle \delta, \nabla_w J \rangle_{\mathcal{H}}. \quad (65)$$

By the Riesz representation theorem, the function $\nabla_w J$ inside the inner product is the unique functional gradient:

$$\nabla_w J(\mathbf{k}, \mathbf{v}) = \frac{e^{\langle \mathbf{k}, \mathbf{q} \rangle}}{D(w)} \langle \nabla_{\hat{\mathbf{y}}} \mathcal{L}, \mathbf{v} - s_w(\mathbf{q}) \rangle. \quad (66)$$

C.4 Constructive Update Analysis

We now analyze the impact of the functional update $w_{t+1} \leftarrow w_t - \eta \nabla_w J$ on the softmax functional $s_{w_{t+1}}(\mathbf{q})$ and demonstrate that it is equivalent to adding a specific key-value tuple to the database. The impact of this update is determined by the changes to the numerator and denominator functionals. For the denominator, the update adds a term:

$$\begin{aligned} \Delta D(\mathbf{q}) &= -\eta \int \nabla_w J(\mathbf{k}, \mathbf{v}) e^{\langle \mathbf{k}, \mathbf{q} \rangle} d\mu(\mathbf{k}, \mathbf{v}) \\ &= -\frac{\eta}{D(w_t)} \int e^{\langle \mathbf{k}, \mathbf{q}_t + \mathbf{q} \rangle} \langle \mathbf{g}_t, \mathbf{v} - \mathbf{s}_t \rangle d\mu(\mathbf{k}, \mathbf{v}), \end{aligned}$$

where $\mathbf{g}_t = \nabla_{\hat{\mathbf{y}}} \mathcal{L}$ and $\mathbf{s}_t = s_{w_t}(\mathbf{q}_t)$. Since the measure μ is a centered Gaussian, $\mathbb{E}_{\mu}[\mathbf{v}] = \mathbf{0}$. Consequently, the term involving $\mathbf{v}$ integrates to zero, leaving only the scalar part:

$$\Delta D(\mathbf{q}) = \frac{\eta \langle \mathbf{g}_t, \mathbf{s}_t \rangle}{D(w_t)} \mathbb{E}_{\mu_k} [e^{\langle \mathbf{k}, \mathbf{q}_t + \mathbf{q} \rangle}].$$

The expectation is the moment generating function of $\mathcal{N}(\mathbf{0}, \Sigma_k)$ evaluated at $\mathbf{q}_t + \mathbf{q}$, which equals $e^{\frac{1}{2} \|\Sigma_k^{1/2}(\mathbf{q}_t + \mathbf{q})\|^2}$. Factoring this, we get a term proportional to $e^{\langle \Sigma_k \mathbf{q}_t, \mathbf{q} \rangle}$. This corresponds precisely to the contribution of a new key $\mathbf{k}_{\text{new}} = \Sigma_k \mathbf{q}_t$ (or $\sigma_k^2 \mathbf{q}_t$ in the isotropic case). Similarly, for the numerator, the update adds a term involving $\mathbf{v}$:

$$\Delta N(\mathbf{q}) = -\frac{\eta}{D(w_t)} \int e^{\langle \mathbf{k}, \mathbf{q}_t + \mathbf{q} \rangle} \langle \mathbf{g}_t, \mathbf{v} - \mathbf{s}_t \rangle \mathbf{v} d\mu(\mathbf{k}, \mathbf{v}).$$

The term $\langle \mathbf{g}_t, -\mathbf{s}_t \rangle \mathbf{v}$ integrates to zero (zero mean). The remaining term involves the second moment $\mathbb{E}_{\mu_v}[\mathbf{v} \mathbf{v}^\top] = \Sigma_v$. Thus:

$$\Delta N(\mathbf{q}) = -\frac{\eta}{D(w_t)} \Sigma_v \mathbf{g}_t \mathbb{E}_{\mu_k} [e^{\langle \mathbf{k}, \mathbf{q}_t + \mathbf{q} \rangle}].$$

Comparing ΔN and ΔD , we see they share the same key kernel $e^{\langle \mathbf{k}_{\text{new}}, \mathbf{q} \rangle}$. The equivalent added tuple $(w_{\text{new}}, \mathbf{k}_{\text{new}}, \mathbf{v}_{\text{new}})$ is thus:

$$\mathbf{k}_{\text{new}} = \Sigma_k \mathbf{q}_t, \quad w_{\text{new}} \propto \eta \langle \mathbf{s}_t, \mathbf{g}_t \rangle, \quad \mathbf{v}_{\text{new}} \propto -\frac{\Sigma_v \mathbf{g}_t}{\langle \mathbf{s}_t, \mathbf{g}_t \rangle}.$$

This confirms that functional gradient descent in $\mathcal{H}_{\sigma_k, \sigma_v}$ is constructively equivalent to growing the database with specific, tractable update rules.

In order to avoid dividing by $\langle \mathbf{s}_t, \mathbf{g}_t \rangle$ which could be small and might introduce numerical instabilities, we can adopt a representation where we store the weighted value vector $\tilde{\mathbf{v}} = w \mathbf{v}$ instead of the value vector $\mathbf{v}$, as in Eq. 8. In this case, the added tuple becomes

$$\mathbf{k}_{\text{new}} = \Sigma_k \mathbf{q}_t, \quad w_{\text{new}} \propto \eta \langle \mathbf{s}_t, \mathbf{g}_t \rangle, \quad \tilde{\mathbf{v}}_{\text{new}} \propto -\eta \Sigma_v \mathbf{g}_t.$$

If $\Sigma_v = \sigma_v^2 \mathbf{I}$ and $\beta = \eta \sigma_v^2$, the weighted value vector becomes

$$\tilde{\mathbf{v}}_{\text{new}} \propto -\beta \mathbf{g}_t.$$

C.5 Negative Weights and Numerical Stability

A critical inspection of the update rule for w_{new} reveals that its sign is determined by the inner product $\delta_t = \langle \mathbf{s}_t, \mathbf{g}_t \rangle$. Since the learning rate η and normalization $D(w_t)$ are positive (assuming initialization with positive mass), w_{new} can be negative if the gradient of the loss forms an obtuse angle with the current prediction vector.

This introduces the concept of “negative attention” or signed measures into the softmax mechanism. Negative weights act to reduce the denominator (partition function), effectively increasing the magnitude of the output vector $s(\mathbf{q})$ in the direction of the value vectors. This occurs when the current prediction $\mathbf{s}_t$ is negatively correlated with the gradient $\mathbf{g}_t$, indicating that gradient descent is attempting to scale up the prediction to minimize the loss.

While considering negative weights might increase the expressive power of our model, they might introduce numerical instabilities as the denominator might become close to 0 if the negative and positive terms cancel.

To ensure robustness, we propose enforcing non-negativity by clipping the new weight:

$$w_{\text{new}} \leftarrow \max(0, w_{\text{new}}). \quad (67)$$

This clipping strategy can be theoretically justified as performing a descent step in a larger, decoupled function space $\mathcal{H}_{\sigma_k, \sigma_v}^2$, where the numerator and denominator weight functions are optimized independently. Refer to Sec. C.6 for further details.

In Sec. C.7, we propose another method for solving the negative weights issue.

We leave the investigation of deriving an update rule that benefits from the potential expressive power of the negative weights while ensuring numerical stability for future research.

C.6 Functional Gradient Descent in the Decoupled Weight Space

In this appendix, we consider the functional gradient descent on the decoupled softmax functional defined in Eq. 68 below. We define the product Hilbert space $\mathcal{H}^2 = \mathcal{H}_{\sigma_k, \sigma_v} \oplus \mathcal{H}_{\sigma_k, \sigma_v}$ containing pairs of weight functions (w_N, w_D) . The decoupled softmax functional is:

$$s_{w_N, w_D}(\mathbf{q}) = \frac{N(w_N)}{D(w_D)} = \frac{\int w_N(\mathbf{k}, \mathbf{v}) e^{\langle \mathbf{k}, \mathbf{q} \rangle} \mathbf{v} d\mu}{\int w_D(\mathbf{k}, \mathbf{v}) e^{\langle \mathbf{k}, \mathbf{q} \rangle} d\mu}. \quad (68)$$

We compute the Fréchet derivatives with respect to w_N and w_D independently. For w_N , the functional is linear in the numerator:

$$\nabla_{w_N} s = \frac{1}{D(w_D)} \nabla_{w_N} N.$$

The gradient of the loss $J(w_N, w_D)$ with respect to w_N is:

$$\nabla_{w_N} J = \frac{1}{D(w_D)} e^{\langle \mathbf{k}, \mathbf{q}_t \rangle} \langle \nabla_{\mathbf{y}} \mathcal{L}, \mathbf{v} \rangle.$$

For w_D , the functional involves the inverse of the denominator:

$$\nabla_{w_D} s = -\frac{N(w_N)}{D(w_D)^2} \nabla_{w_D} D = -\frac{s_{w_N, w_D}}{D(w_D)} \nabla_{w_D} D.$$

The gradient of the loss with respect to w_D is:

$$\nabla_{w_D} J = -\frac{1}{D(w_D)} e^{\langle \mathbf{k}, \mathbf{q}_t \rangle} \langle \nabla_{\mathbf{y}} \mathcal{L}, s_{w_N, w_D}(\mathbf{q}_t) \rangle.$$

We now map these updates to the growing database formulation. Updating w_N adds a term $\Delta w_N \propto -\frac{1}{D} \langle \mathbf{g}_t, \mathbf{v} \rangle e^{\langle \mathbf{k}, \mathbf{q}_t \rangle}$. In the “weighted value vector” representation (where we store $\tilde{\mathbf{v}}$ instead of $\mathbf{v}$), adding a tuple corresponds to adding a basis function to w_N . Specifically, if we add $(\tilde{\mathbf{v}}_{\text{new}}, \mathbf{k}_{\text{new}})$, we modify the numerator density by a term proportional to $e^{\langle \mathbf{k}, \Sigma_k^{-1} \mathbf{k}_{\text{new}} \rangle} \langle \mathbf{v}, \Sigma_v^{-1} \tilde{\mathbf{v}}_{\text{new}} \rangle$. Matching terms, we find $\mathbf{k}_{\text{new}} = \Sigma_k \mathbf{q}_t$ and $\tilde{\mathbf{v}}_{\text{new}} \propto -\eta \Sigma_v \mathbf{g}_t$. This matches the numerator update derived in the coupled case.

Updating w_D adds a term $\Delta w_D \propto \frac{1}{D} \langle \mathbf{g}_t, \mathbf{s}_t \rangle e^{\langle \mathbf{k}, \mathbf{q}_t \rangle}$. This corresponds to adding a basis function to the denominator weight density w_D . In the database, this means adding a tuple $(w_{\text{new}}, \mathbf{k}_{\text{new}})$. Matching terms, we find $\mathbf{k}_{\text{new}} = \Sigma_k \mathbf{q}_t$ and $w_{\text{new}} \propto \eta \langle \mathbf{s}_t, \mathbf{g}_t \rangle$. This matches the denominator update derived in the coupled case.

Therefore, the coupled update rule derived in Appendix A is a special case of the decoupled update where we enforce $w_N = w_D = w$ initially but allow them to evolve. Crucially, since the optimization of w_D is independent of w_N in this formulation, applying a constraint $w_D \geq 0$ (via clipping $w_{\text{new}} = \max(0, w_{\text{new}})$) corresponds to a valid Descent step in $\mathcal{H}_{\sigma_k, \sigma_v} \oplus \mathcal{H}_{\sigma_k, \sigma_v}$ which would be positively correlated with the gradient direction. This ensures numerical stability while maintaining the numerator’s ability to reduce error.

C.7 Gauge-Invariance: Another Method for Solving the Negative Weights Issue

As established in Sec. C.5, the constructive functional gradient update for the scalar weight of a newly added weighted key-value tuple is given by:

$$w_{\text{new}} \propto \eta \langle \mathbf{s}_t, \mathbf{g}_t \rangle, \quad (69)$$

where $\mathbf{s}_t = s_{w_t}(\mathbf{q}_t)$ is the current prediction and $\mathbf{g}_t = \nabla_{\mathbf{y}} \mathcal{L}$ is the error signal. When the gradient forms an obtuse angle with the current prediction vector (i.e., $\langle \mathbf{s}_t, \mathbf{g}_t \rangle < 0$), this rule mandates appending a *negative weight* ($w_{\text{new}} < 0$). While the non-negativity constraint can be enforced heuristically via clipping $w_{\text{new}} \leftarrow \max(0, w_{\text{new}})$ (theoretically grounded in the decoupled weight space of Sec. C.6), clipping discards the denominator update entirely whenever $\langle \mathbf{s}_t, \mathbf{g}_t \rangle < 0$.

In this appendix, we develop an alternative, geometrically principled resolution to the negative weight obstruction: an **augmented representation with gauge invariance**. By introducing a global bias degree of freedom, we show that the underlying function space possesses an exact translation symmetry in value space. By dynamically selecting an appropriate coordinate frame (a “virtual gauge”) at each training step, we guarantee that the required weight update is *strictly positive* ($w_{\text{new}} > 0$) without clipping, while preserving the locality of the database update.

C.7.1 Augmented Functional and Gauge Symmetry

We introduce an auxiliary global bias degree of freedom $\mathbf{b} \in \mathbb{R}^{d_{\text{out}}}$ into both the continuous and discrete weighted softmax functionals.

Definition C.1 (Augmented Softmax Functionals). Let $\mathbf{b} \in \mathbb{R}^{d_{\text{out}}}$ be a bias vector.

1. For a functional weight density $w \in \mathcal{H}_{\sigma_k, \sigma_v}$, the continuous augmented softmax output $s_{w, \mathbf{b}} : \mathbb{R}^{d_{\text{in}}} \rightarrow \mathbb{R}^{d_{\text{out}}}$ is defined as:

$$s_{w, \mathbf{b}}(\mathbf{q}) := s_w(\mathbf{q}) - \mathbf{b} = \frac{\int_{\mathbb{R}^{d_{\text{in}}} \times \mathbb{R}^{d_{\text{out}}}} w(\mathbf{k}, \mathbf{v}) e^{\langle \mathbf{k}, \mathbf{q} \rangle} \mathbf{v} d\mu(\mathbf{k}, \mathbf{v})}{\int_{\mathbb{R}^{d_{\text{in}}} \times \mathbb{R}^{d_{\text{out}}}} w(\mathbf{k}, \mathbf{v}) e^{\langle \mathbf{k}, \mathbf{q} \rangle} d\mu(\mathbf{k}, \mathbf{v})} - \mathbf{b}. \quad (70)$$

2. For a discrete database $\mathcal{D} = \{(w_i, \mathbf{k}_i, \tilde{\mathbf{v}}_i)\}_{i=1}^N$ of weighted key-value tuples with $\tilde{\mathbf{v}}_i = w_i \mathbf{v}_i$, the discrete augmented retrieval functional is defined as:

$$s_{\mathcal{D}, \mathbf{b}}(\mathbf{q}) := s_{\mathcal{D}}(\mathbf{q}) - \mathbf{b} = \frac{\sum_{i=1}^N e^{\langle \mathbf{k}_i, \mathbf{q} \rangle} \tilde{\mathbf{v}}_i}{\sum_{i=1}^N w_i e^{\langle \mathbf{k}_i, \mathbf{q} \rangle}} - \mathbf{b}. \quad (71)$$

The standard model corresponds to a bias that is set to zero: $s_{\mathcal{D}}(\mathbf{q}) = s_{\mathcal{D}, \mathbf{0}}(\mathbf{q})$.

We now define a transformation of the representation under a spatial translation in value space.

Definition C.2 (Shifted Density and Database). Let $\boldsymbol{\xi} \in \mathbb{R}^{d_{\text{out}}}$ be an arbitrary shift vector.

- We define the shifted weight density $w_{\boldsymbol{\xi}}$ by translating the value coordinate:

$$w_{\boldsymbol{\xi}}(\mathbf{k}, \mathbf{v}) := w(\mathbf{k}, \mathbf{v} - \boldsymbol{\xi}) \cdot \exp\left(\frac{2\langle \boldsymbol{\xi}, \mathbf{v} \rangle - \|\boldsymbol{\xi}\|^2}{2\sigma_v^2}\right). \quad (72)$$

- In the discrete representation, we shift the value vectors as $\mathbf{v}_i \mapsto \mathbf{v}_i + \boldsymbol{\xi}$ which implies that the stored weighted values shift as $\tilde{\mathbf{v}}_i \mapsto w_i(\mathbf{v}_i + \boldsymbol{\xi}) = \tilde{\mathbf{v}}_i + w_i\boldsymbol{\xi}$. Hence, we define the shifted database $\mathcal{D}_\xi$ as:

$$\mathcal{D}_\xi := \{(w_i, \mathbf{k}_i, \tilde{\mathbf{v}}_i + w_i\boldsymbol{\xi})\}_{i=1}^N. \quad (73)$$

Proposition C.3 (Gauge Invariance). *For any density $w \in \mathcal{H}_{\sigma_k, \sigma_v}$, discrete database $\mathcal{D}$, bias vector $\mathbf{b} \in \mathbb{R}^{d_{out}}$, and shift vector $\boldsymbol{\xi} \in \mathbb{R}^{d_{out}}$, the augmented functional is invariant under simultaneous shifts of the values and the bias:*

$$s_{w_\xi, \mathbf{b}+\boldsymbol{\xi}}(\mathbf{q}) = s_{w, \mathbf{b}}(\mathbf{q}) \quad \text{and} \quad s_{\mathcal{D}_\xi, \mathbf{b}+\boldsymbol{\xi}}(\mathbf{q}) = s_{\mathcal{D}, \mathbf{b}}(\mathbf{q}). \quad (74)$$

In particular, any augmented state $(\mathcal{D}, \mathbf{b})$ is gauge-equivalent to a canonical zero-bias state:

$$s_{\mathcal{D}, \mathbf{b}}(\mathbf{q}) = s_{\mathcal{D}-\mathbf{b}, \mathbf{0}}(\mathbf{q}). \quad (75)$$

Proof. For the continuous case, note that

$$\begin{aligned} s_{w_\xi}(\mathbf{q}) &= \frac{\int_{\mathbb{R}^{d_{in}} \times \mathbb{R}^{d_{out}}} w_\xi(\mathbf{k}, \mathbf{v}) e^{\langle \mathbf{k}, \mathbf{q} \rangle} \mathbf{v} d\mu(\mathbf{k}, \mathbf{v})}{\int_{\mathbb{R}^{d_{in}} \times \mathbb{R}^{d_{out}}} w_\xi(\mathbf{k}, \mathbf{v}) e^{\langle \mathbf{k}, \mathbf{q} \rangle} d\mu(\mathbf{k}, \mathbf{v})} \\ &= \frac{\int_{\mathbb{R}^{d_{in}} \times \mathbb{R}^{d_{out}}} w(\mathbf{k}, \mathbf{v} - \boldsymbol{\xi}) \cdot \exp\left(\frac{2\langle \boldsymbol{\xi}, \mathbf{v} \rangle - \|\boldsymbol{\xi}\|^2}{2\sigma_v^2}\right) e^{\langle \mathbf{k}, \mathbf{q} \rangle} \mathbf{v} \exp\left(-\frac{\|\mathbf{k}\|^2}{2\sigma_k^2} - \frac{\|\mathbf{v}\|^2}{2\sigma_v^2}\right) d\mathbf{k}d\mathbf{v}}{\int_{\mathbb{R}^{d_{in}} \times \mathbb{R}^{d_{out}}} w(\mathbf{k}, \mathbf{v} - \boldsymbol{\xi}) \cdot \exp\left(\frac{2\langle \boldsymbol{\xi}, \mathbf{v} \rangle - \|\boldsymbol{\xi}\|^2}{2\sigma_v^2}\right) e^{\langle \mathbf{k}, \mathbf{q} \rangle} \exp\left(-\frac{\|\mathbf{k}\|^2}{2\sigma_k^2} - \frac{\|\mathbf{v}\|^2}{2\sigma_v^2}\right) d\mathbf{k}d\mathbf{v}} \\ &= \frac{\int_{\mathbb{R}^{d_{in}} \times \mathbb{R}^{d_{out}}} w(\mathbf{k}, \mathbf{v} - \boldsymbol{\xi}) e^{\langle \mathbf{k}, \mathbf{q} \rangle} \mathbf{v} \exp\left(-\frac{\|\mathbf{k}\|^2}{2\sigma_k^2} - \frac{\|\mathbf{v} - \boldsymbol{\xi}\|^2}{2\sigma_v^2}\right) d\mathbf{k}d\mathbf{v}}{\int_{\mathbb{R}^{d_{in}} \times \mathbb{R}^{d_{out}}} w(\mathbf{k}, \mathbf{v} - \boldsymbol{\xi}) e^{\langle \mathbf{k}, \mathbf{q} \rangle} \exp\left(-\frac{\|\mathbf{k}\|^2}{2\sigma_k^2} - \frac{\|\mathbf{v} - \boldsymbol{\xi}\|^2}{2\sigma_v^2}\right) d\mathbf{k}d\mathbf{v}} \\ &= \frac{\int_{\mathbb{R}^{d_{in}} \times \mathbb{R}^{d_{out}}} w(\mathbf{k}, \mathbf{v}) e^{\langle \mathbf{k}, \mathbf{q} \rangle} (\mathbf{v} + \boldsymbol{\xi}) \exp\left(-\frac{\|\mathbf{k}\|^2}{2\sigma_k^2} - \frac{\|\mathbf{v}\|^2}{2\sigma_v^2}\right) d\mathbf{k}d\mathbf{v}}{\int_{\mathbb{R}^{d_{in}} \times \mathbb{R}^{d_{out}}} w(\mathbf{k}, \mathbf{v}) e^{\langle \mathbf{k}, \mathbf{q} \rangle} \exp\left(-\frac{\|\mathbf{k}\|^2}{2\sigma_k^2} - \frac{\|\mathbf{v}\|^2}{2\sigma_v^2}\right) d\mathbf{k}d\mathbf{v}} \\ &= \frac{\int_{\mathbb{R}^{d_{in}} \times \mathbb{R}^{d_{out}}} w(\mathbf{k}, \mathbf{v}) e^{\langle \mathbf{k}, \mathbf{q} \rangle} (\mathbf{v} + \boldsymbol{\xi}) d\mu(\mathbf{k}, \mathbf{v})}{\int_{\mathbb{R}^{d_{in}} \times \mathbb{R}^{d_{out}}} w(\mathbf{k}, \mathbf{v}) e^{\langle \mathbf{k}, \mathbf{q} \rangle} d\mu(\mathbf{k}, \mathbf{v})} \\ &= s_w(\mathbf{q}) + \boldsymbol{\xi}. \end{aligned}$$

Evaluating the augmented functional with shifted bias $\mathbf{b} + \boldsymbol{\xi}$, we get

$$s_{w_\xi, \mathbf{b}+\boldsymbol{\xi}}(\mathbf{q}) = s_{w_\xi}(\mathbf{q}) - (\mathbf{b} + \boldsymbol{\xi}) = (s_w(\mathbf{q}) + \boldsymbol{\xi}) - (\mathbf{b} + \boldsymbol{\xi}) = s_w(\mathbf{q}) - \mathbf{b} = s_{w, \mathbf{b}}(\mathbf{q}). \quad (76)$$

Similarly, for the discrete case, we have:

$$\begin{aligned} s_{\mathcal{D}_\xi}(\mathbf{q}) &= \frac{\sum_{i=1}^N e^{\langle \mathbf{k}_i, \mathbf{q} \rangle} (\tilde{\mathbf{v}}_i + w_i\boldsymbol{\xi})}{\sum_{i=1}^N w_i e^{\langle \mathbf{k}_i, \mathbf{q} \rangle}} \\ &= \frac{\sum_{i=1}^N e^{\langle \mathbf{k}_i, \mathbf{q} \rangle} \tilde{\mathbf{v}}_i}{\sum_{i=1}^N w_i e^{\langle \mathbf{k}_i, \mathbf{q} \rangle}} + \boldsymbol{\xi} \frac{\sum_{i=1}^N w_i e^{\langle \mathbf{k}_i, \mathbf{q} \rangle}}{\sum_{i=1}^N w_i e^{\langle \mathbf{k}_i, \mathbf{q} \rangle}} \\ &= s_{\mathcal{D}}(\mathbf{q}) + \boldsymbol{\xi}. \end{aligned} \quad (77)$$

Evaluating the augmented functional with shifted bias $\mathbf{b} + \boldsymbol{\xi}$, we get

$$s_{\mathcal{D}_\xi, \mathbf{b}+\boldsymbol{\xi}}(\mathbf{q}) = s_{\mathcal{D}_\xi}(\mathbf{q}) - (\mathbf{b} + \boldsymbol{\xi}) = (s_{\mathcal{D}}(\mathbf{q}) + \boldsymbol{\xi}) - (\mathbf{b} + \boldsymbol{\xi}) = s_{\mathcal{D}}(\mathbf{q}) - \mathbf{b} = s_{\mathcal{D}, \mathbf{b}}(\mathbf{q}). \quad (78)$$

Choosing $\boldsymbol{\xi} = -\mathbf{b}$ yields $s_{\mathcal{D}, \mathbf{b}}(\mathbf{q}) = s_{\mathcal{D}-\mathbf{b}, \mathbf{0}}(\mathbf{q})$, completing the proof. $\square$

C.7.2 Gauge Fixing for Strictly Positive Weight Updates

Suppose at step t the model is stored in the canonical zero-bias frame $\mathcal{D}^{(t)}$ with $\mathbf{b} = \mathbf{0}$, producing prediction $\mathbf{s}_t = s_{\mathcal{D}^{(t)}}(\mathbf{q}_t)$ on query $\mathbf{q}_t$ and loss gradient $\mathbf{g}_t = \nabla_{\mathbf{y}} \mathcal{L}(\mathbf{s}_t, \mathbf{y}_t)$.

Instead of computing the functional gradient in the canonical frame (which risks $w_{\text{new}} < 0$), we perform a gauge transformation to view the database in a **virtual gauge** with bias $\mathbf{b}_t \in \mathbb{R}^{d_{out}}$. By

Theorem C.3, mapping the canonical zero-bias state to a frame with bias $\mathbf{b}_t$ requires shifting the values by $\boldsymbol{\xi} = \mathbf{b}_t$, resulting in the shifted virtual database:

$$\mathcal{D}_{\mathbf{b}_t}^{(t)} = \{(w_i, \mathbf{k}_i, \tilde{\mathbf{v}}_i + w_i \mathbf{b}_t)\}_{i=1}^N. \quad (79)$$

Recall that $s_{\mathcal{D}_{\mathbf{b}_t}^{(t)}, \mathbf{b}_t}(\mathbf{q}_t) = s_{\mathcal{D}_{\mathbf{b}_t}^{(t)}}(\mathbf{q}_t) - \mathbf{b}_t = s_{\mathcal{D}^{(t)}}(\mathbf{q}_t) + \mathbf{b}_t - \mathbf{b}_t = s_{\mathcal{D}^{(t)}}(\mathbf{q}_t)$.

Now consider adding a tuple $(w_{\text{new}}, \mathbf{k}_{\text{new}}, \tilde{\mathbf{v}}'_{\text{new}})$ to $\mathcal{D}_{\mathbf{b}_t}^{(t)}$ in this frame. As derived in Sec. C.4, the functional gradient descent update on the weight density w yields a scalar mass update proportional to the projection of the gradient onto the effective prediction relative to the frame bias:

$$w_{\text{new}} = \eta \left\langle \mathbf{g}_t, s_{\mathcal{D}_{\mathbf{b}_t}^{(t)}}(\mathbf{q}_t) \right\rangle = \eta \left\langle \mathbf{g}_t, s_{\mathcal{D}^{(t)}}(\mathbf{q}_t) + \mathbf{b}_t \right\rangle = \eta \langle \mathbf{g}_t, \mathbf{s}_t + \mathbf{b}_t \rangle. \quad (80)$$

Because the gauge choice $\mathbf{b}_t$ is a free degree of freedom that leaves the function invariant, we can **fix the gauge dynamically** at each step to ensure that the inner product in equation 80 is strictly positive. Specifically, we choose:

$$\mathbf{b}_t = (-\langle \mathbf{s}_t, \mathbf{g}_t \rangle / \|\mathbf{g}_t\|^2 + \alpha_t) \mathbf{g}_t, \quad (81)$$

where $\alpha_t > 0$ is a scalar gauge parameter. Substituting equation 81 into equation 80 yields:

$$w_{\text{new}} = \eta \left\langle \mathbf{g}_t, \mathbf{s}_t + (-\langle \mathbf{s}_t, \mathbf{g}_t \rangle / \|\mathbf{g}_t\|^2 + \alpha_t) \mathbf{g}_t \right\rangle = \eta \alpha_t \|\mathbf{g}_t\|^2 > 0. \quad (82)$$

Thus, for any non-zero error gradient $\mathbf{g}_t$, the new weight is **guaranteed to be strictly positive**.

The target weighted value vector $\tilde{\mathbf{v}}'_{\text{new}}$ in the virtual frame is directed along the negative error gradient, following standard functional descent on the numerator functional:

$$\tilde{\mathbf{v}}'_{\text{new}} = -\beta \mathbf{g}_t, \quad (83)$$

where $\beta > 0$ is the value learning rate. The key is set as usual to $\mathbf{k}_{\text{new}} = \Sigma_k \mathbf{q}_t = \sigma_k^2 \mathbf{q}_t$. Appending this tuple to $\mathcal{D}_{\mathbf{b}_t}^{(t)}$ yields the preliminary database $\mathcal{D}^{\text{prel}, (t+1)}$ in the virtual frame:

$$\mathcal{D}^{\text{prel}, (t+1)} := \mathcal{D}_{\mathbf{b}_t}^{(t)} \cup \{(w_{\text{new}}, \mathbf{k}_{\text{new}}, \tilde{\mathbf{v}}'_{\text{new}})\}.$$

C.7.3 Canonical Return and Locality of the Database Update

To maintain a standard, zero-bias database across training steps, we must transform $\mathcal{D}^{\text{prel}, (t+1)}$ back to the canonical gauge ($\mathbf{b} = \mathbf{0}$). Shifting the bias from $\mathbf{b}_t$ back to $\mathbf{0}$ corresponds to applying the shift $\boldsymbol{\xi} = -\mathbf{b}_t$ to all values.

A naive implementation might suggest that shifting values by $-\mathbf{b}_t$ requires modifying every existing entry in the database. Remarkably, this is not the case: the gauge transformation yields a local update.

Theorem C.4 (Locality of the Gauge-Invariant Update). *The transition from the canonical database $\mathcal{D}^{(t)}$ to $\mathcal{D}^{(t+1)}$ consists solely of appending a single new tuple:*

$$\mathcal{D}^{(t+1)} = \mathcal{D}^{(t)} \cup \{(w_{\text{new}}, \mathbf{k}_{\text{new}}, \tilde{\mathbf{v}}_{\text{final}})\}, \quad (84)$$

where all previously stored tuples remain completely unaltered, and the canonical (weighted) value vector of the new tuple is given by:

$$\tilde{\mathbf{v}}_{\text{final}} = (-w_{\text{new}} \alpha_t - \beta + \eta \alpha_t \langle \mathbf{s}_t, \mathbf{g}_t \rangle) \mathbf{g}_t. \quad (85)$$

Proof. Let $(w_i, \mathbf{k}_i, \tilde{\mathbf{v}}_i) \in \mathcal{D}^{(t)}$ be an arbitrary pre-existing tuple.

1. **Forward transformation to virtual gauge:** Under shift $\boldsymbol{\xi} = +\mathbf{b}_t$, the weighted value becomes $\tilde{\mathbf{v}}'_i = \tilde{\mathbf{v}}_i + w_i \mathbf{b}_t$.
2. **Atom insertion:** The new atom $(w_{\text{new}}, \mathbf{k}_{\text{new}}, \tilde{\mathbf{v}}'_{\text{new}})$ is added. Existing tuples are unaffected.
3. **Return transformation to canonical gauge:** Shifting the bias by $-\mathbf{b}_t$ transforms all weighted values via $\tilde{\mathbf{v}} \mapsto \tilde{\mathbf{v}} - w \mathbf{b}_t$. For any existing tuple i :

$$\tilde{\mathbf{v}}''_i = \tilde{\mathbf{v}}'_i - w_i \mathbf{b}_t = (\tilde{\mathbf{v}}_i + w_i \mathbf{b}_t) - w_i \mathbf{b}_t = \tilde{\mathbf{v}}_i. \quad (86)$$

All previously stored tuples return identically to their initial values.

For the newly added atom, its weighted value in the virtual frame was $\tilde{\mathbf{v}}'_{\text{new}} = -\beta \mathbf{g}_t$. Shifting it to the canonical frame yields:

$$\begin{aligned}\tilde{\mathbf{v}}_{\text{final}} &= \tilde{\mathbf{v}}'_{\text{new}} - w_{\text{new}} \mathbf{b}_t \\ &= -\beta \mathbf{g}_t - w_{\text{new}} (-\langle \mathbf{s}_t, \mathbf{g}_t \rangle / \|\mathbf{g}_t\|^2 + \alpha_t) \mathbf{g}_t \\ &= \left(-w_{\text{new}} \alpha_t - \beta + w_{\text{new}} \frac{\langle \mathbf{s}_t, \mathbf{g}_t \rangle}{\|\mathbf{g}_t\|^2} \right) \mathbf{g}_t\end{aligned}\tag{87}$$

$$= \left(-w_{\text{new}} \alpha_t - \beta + \eta \alpha_t \|\mathbf{g}_t\|^2 \frac{\langle \mathbf{s}_t, \mathbf{g}_t \rangle}{\|\mathbf{g}_t\|^2} \right) \mathbf{g}_t\tag{88}$$

$$= (-w_{\text{new}} \alpha_t - \beta + \eta \alpha_t \langle \mathbf{s}_t, \mathbf{g}_t \rangle) \mathbf{g}_t.\tag{89}$$

Thus, no historical data points are modified, and the update is strictly local. $\square$

C.7.4 Algorithm Summary

The complete procedure is formalized in Algorithm 1. As mentioned in Remark 3.1, we store $\mathbf{k}_{\text{new}} \leftarrow \mathbf{q}_t$ in the database instead of $\mathbf{k}_{\text{new}} \leftarrow \sigma_k^2 \mathbf{q}_t$.

Algorithm 1 Gauge-Invariant Weighted Softmax Growing Step

- 1: **Input:** Current database $\mathcal{D}^{(t)}$, input query $\mathbf{q}_t$, loss gradient $\mathbf{g}_t = \nabla_{\mathbf{y}} \mathcal{L}(\mathbf{s}_t, \mathbf{y}_t)$
 - 2: **Hyperparameters:** Weight learning rate $\eta > 0$, value learning rate $\beta > 0$, gauge parameter $\alpha_t > 0$, temperature T
 - 3: **1. Compute prediction:**
 - 4: $\mathbf{s}_t \leftarrow \frac{\sum_{(w, \mathbf{k}, \tilde{\mathbf{v}}) \in \mathcal{D}^{(t)}} e^{\frac{1}{T\sqrt{d_{\text{in}}}} \langle \mathbf{k}, \mathbf{q}_t \rangle} \tilde{\mathbf{v}}}{\sum_{(w, \mathbf{k}, \tilde{\mathbf{v}}) \in \mathcal{D}^{(t)}} e^{\frac{1}{T\sqrt{d_{\text{in}}}} \langle \mathbf{k}, \mathbf{q}_t \rangle} w}$
 - 5: **2. Compute strictly positive weight update:**
 - 6: $w_{\text{new}} \leftarrow \eta \alpha_t \|\mathbf{g}_t\|^2$
 - 7: **3. Compute canonical weighted value:**
 - 8: $\tilde{\mathbf{v}}_{\text{final}} \leftarrow (-w_{\text{new}} \alpha_t - \beta + \eta \alpha_t \langle \mathbf{s}_t, \mathbf{g}_t \rangle) \mathbf{g}_t$
 - 9: **4. Set new key:**
 - 10: $\mathbf{k}_{\text{new}} \leftarrow \mathbf{q}_t$
 - 11: **5. Append new tuple to database:**
 - 12: $\mathcal{D}^{(t+1)} \leftarrow \mathcal{D}^{(t)} \cup \{(w_{\text{new}}, \mathbf{k}_{\text{new}}, \tilde{\mathbf{v}}_{\text{final}})\}$
-

C.7.5 Hyperparameter Choice that Yields Changes in the Order of the Gradient Length

The weight of the added tuple is $w_{\text{new}} = \eta \alpha_t \|\mathbf{g}_t\|^2$ which is quadratic in the length of the gradient. In order to ensure that we add updates that are in the order of the gradient length, we can set $\alpha_t := \frac{\alpha}{\|\mathbf{g}_t\|}$. In this case, we get

$$w_{\text{new}} = \eta \alpha \|\mathbf{g}_t\|,$$

and

$$\begin{aligned}\tilde{\mathbf{v}}_{\text{final}} &= (-w_{\text{new}} \alpha_t - \beta + \eta \alpha_t \langle \mathbf{s}_t, \mathbf{g}_t \rangle) \mathbf{g}_t \\ &= (-\eta \alpha_t \|\mathbf{g}_t\|^2 \alpha_t - \beta + \eta \alpha_t \langle \mathbf{s}_t, \mathbf{g}_t \rangle) \mathbf{g}_t \\ &= \left(-\eta \frac{\alpha}{\|\mathbf{g}_t\|} \|\mathbf{g}_t\|^2 \frac{\alpha}{\|\mathbf{g}_t\|} - \beta + \eta \frac{\alpha}{\|\mathbf{g}_t\|} \langle \mathbf{s}_t, \mathbf{g}_t \rangle \right) \mathbf{g}_t \\ &= \left(-\eta \alpha^2 - \beta + \eta \frac{\alpha}{\|\mathbf{g}_t\|} \langle \mathbf{s}_t, \mathbf{g}_t \rangle \right) \mathbf{g}_t.\end{aligned}$$

This yields the following algorithm.

Algorithm 2 Gauge-Invariant Weighted Softmax Learning with Updates in the Order of the Gradient Length

- 1: **Input:** Current database $\mathcal{D}^{(t)}$, input query $\mathbf{q}_t$, loss gradient $\mathbf{g}_t = \nabla_{\mathbf{y}} \mathcal{L}(\mathbf{s}_t, \mathbf{y}_t)$
 - 2: **Hyperparameters:** Weight learning rate $\eta > 0$, value learning rate $\beta > 0$, gauge parameter $\alpha > 0$, temperature T
 - 3: **1. Compute prediction:**
 - 4:
$$\mathbf{s}_t \leftarrow \frac{\sum_{(w, \mathbf{k}, \tilde{\mathbf{v}}) \in \mathcal{D}^{(t)}} e^{\frac{1}{T\sqrt{d_{\text{in}}}} \langle \mathbf{k}, \mathbf{q}_t \rangle} \tilde{\mathbf{v}}}{\sum_{(w, \mathbf{k}, \tilde{\mathbf{v}}) \in \mathcal{D}^{(t)}} e^{\frac{1}{T\sqrt{d_{\text{in}}}} \langle \mathbf{k}, \mathbf{q}_t \rangle} w}$$
 - 5: **2. Compute strictly positive weight update:**
 - 6: $w_{\text{new}} \leftarrow \eta \alpha \|\mathbf{g}_t\|$
 - 7: **3. Compute canonical weighted value:**
 - 8:
$$\tilde{\mathbf{v}}_{\text{final}} \leftarrow \left(-\eta \alpha^2 - \beta + \eta \frac{\alpha}{\|\mathbf{g}_t\|} \langle \mathbf{s}_t, \mathbf{g}_t \rangle \right) \mathbf{g}_t$$
 - 9: **4. Set new key:**
 - 10: $\mathbf{k}_{\text{new}} \leftarrow \mathbf{q}_t$
 - 11: **5. Append new tuple to database:**
 - 12: $\mathcal{D}^{(t+1)} \leftarrow \mathcal{D}^{(t)} \cup \{(w_{\text{new}}, \mathbf{k}_{\text{new}}, \tilde{\mathbf{v}}_{\text{final}})\}$
-

D Optimization Dynamics in the Growing Database

Standard deep learning optimization techniques such as weight decay and momentum can be naturally incorporated into the growing database framework. Weight decay, which corresponds to L_2 regularization in the functional space, translates to a simple uniform decay of the scalar weights w and weighted values $\tilde{\mathbf{v}}$ of all stored key-value pairs at each step. This mechanism not only regularizes the model but also introduces a “forgetting” factor where older memories gradually fade in influence. As we show in Sec. D.1, weight-decay corresponds to the update rule

$$\begin{aligned} \mathcal{D}_t = & \{((1 - \gamma)w, \mathbf{k}, (1 - \gamma)\tilde{\mathbf{v}}) \mid (w, \mathbf{k}, \tilde{\mathbf{v}}) \in \mathcal{D}_{t-1}\} \\ & \cup \{(w_{\text{new}}, \mathbf{k}_{\text{new}}, \tilde{\mathbf{v}}_{\text{new}})\}. \end{aligned} \quad (90)$$

This means that the weights of old tuples decay exponentially with training steps, thereby providing a principled path towards “compression” of the database: Old tuples with negligible weights can be pruned with minimal impact on the function, potentially allowing the database to be maintained at a constant size.

Momentum, typically implemented by maintaining a velocity vector, would naively require doubling the storage (storing a velocity database alongside the parameter database). However, we derive a more efficient formulation where momentum is realized analytically. By storing the creation timestamp t_i of each key, we can compute the effective momentum contribution of a key during the forward pass via a time-dependent scaling coefficient:

$$c(t, t_i) = \frac{1 - \mu^{t-t_i+1}}{1 - \mu}, \quad (91)$$

where μ is the momentum factor. As we show in Sec. D.2, this allows the growing network to benefit from accelerated convergence without the storage overhead of explicit velocity buffers.

One can also combine both approaches. The resulting algorithm applies the geometric decay to the stored weights at every step to handle regularization, while applying the time-dependent momentum coefficient during the retrieval operation to simulate the accumulation of gradients. We provide detailed derivations of these dynamics in Sec. D.3.

In this appendix, we derive the update rules for weight decay and momentum within the growing softmax framework. We assume the decoupled representation where the database stores $(w, \mathbf{k}, \tilde{\mathbf{v}})$, and the functions are $w_D(\mathbf{k}) = \sum w_i e^{\langle \mathbf{k}, \mathbf{k}_i \rangle}$ and $w_N(\mathbf{k}) = \sum \tilde{v}_i e^{\langle \mathbf{k}, \mathbf{k}_i \rangle}$. Note that here we use $\tilde{\mathbf{v}}$ to denote the weighted value vector, consistent with the main text notation.

D.1 Weight Decay

Weight decay corresponds to minimizing a regularized objective:

$$J_{reg}(w_N, w_D) = J(w_N, w_D) + \frac{\gamma}{2\eta}(\|w_N\|_{\mathcal{H}}^2 + \|w_D\|_{\mathcal{H}}^2).$$

Here, we adopt the “decoupled weight” formulation of Sec. C.6. Nevertheless, following a “coupled weight” formulation as in equation 5 yields similar results.

The functional gradient descent update for w_D becomes:

$$w_{D,t} \leftarrow w_{D,t-1} - \eta(\nabla_{w_D} J + \frac{\gamma}{\eta} w_{D,t-1}) = (1 - \gamma)w_{D,t-1} - \eta \nabla_{w_D} J.$$

Similarly for the numerator function w_N :

$$w_{N,t} \leftarrow (1 - \gamma)w_{N,t-1} - \eta \nabla_{w_N} J.$$

In the database representation, the function $w_{D,t-1}$ is represented by the sum of existing kernels with weights w_i . Multiplying the function by $(1 - \gamma)$ is equivalent to multiplying every coefficient w_i in the database by $(1 - \gamma)$. The same logic applies to the numerator function and its coefficients $\tilde{v}_i$. Thus, the update rule for the database $\mathcal{D}_{t-1}$ is:

$$\mathcal{D}_t = \{((1 - \gamma)w, \mathbf{k}, (1 - \gamma)\tilde{\mathbf{v}}) \mid (w, \mathbf{k}, \tilde{\mathbf{v}}) \in \mathcal{D}_{t-1}\} \cup \{(w_{\text{new}}, \mathbf{k}_{\text{new}}, \tilde{\mathbf{v}}_{\text{new}})\},$$

where the new tuple is derived from the gradients $-\eta \nabla_{w_D} J$ and $-\eta \nabla_{w_N} J$ as in Sec. C.4.

This operation implies that the weight of a tuple added at time t_0 decays geometrically as $(1 - \gamma)^{t-t_0}$ at step t . Consequently, older tuples will eventually have weights strictly bounded by some small ϵ . This suggests a natural pruning strategy: removing tuples where $|w| < \epsilon$ and $\|\tilde{\mathbf{v}}\| < \epsilon$ results in a negligible perturbation to the softmax function output. By setting a threshold, one can effectively bound the maximum size of the database, converting the growing nonparametric model into a fixed-capacity model with a dynamic memory.

D.2 Momentum

Momentum accelerates optimization by accumulating a velocity function $\nu \in \mathcal{H}$. We apply momentum to the gradient updates for both w_N and w_D . Let’s focus on w_D (the derivation for w_N is identical). The update rules are:

$$\nu_t = \mu \nu_{t-1} + \nabla_{w_D} J_t, \quad w_{D,t} = w_{D,t-1} - \eta \nu_t.$$

Unrolling the recursion, the weight function at step t can be expressed as a linear combination of all past gradients:

$$w_{D,t} = w_{D,0} - \eta \sum_{j=1}^t \left(\sum_{k=0}^{t-j} \mu^k \right) \nabla_{w_D} J_j = w_{D,0} - \eta \sum_{j=1}^t \frac{1 - \mu^{t-j+1}}{1 - \mu} \nabla_{w_D} J_j.$$

Recall that the gradient $\nabla_{w_D} J_j$ at step j corresponds to adding a new kernel centered at $\mathbf{k}_j = \sigma^2 \mathbf{q}_j$ with a specific weight Δw_j . The unrolled equation implies that the accumulated weight for the kernel at $\mathbf{k}_j$ at current time t is simply the initial update weight Δw_j scaled by the coefficient:

$$c(t, t_j) = \frac{1 - \mu^{t-t_j+1}}{1 - \mu}.$$

Therefore, we can implement momentum without storing the velocity function ν_t explicitly. Instead, we store the creation time t_j for each tuple. During the retrieval (forward pass) at current time t , the contribution of the j -th tuple to the denominator is scaled by $c(t, t_j)$. The same scaling applies to the weighted value vector $\tilde{\mathbf{v}}_j$ in the numerator.

D.3 Combined Dynamics

In this section, we provide a rigorous derivation of the update rule when both weight decay and momentum are active. We adopt the **decoupled weight decay** formulation [39], which is the standard for modern deep learning optimizers like AdamW and SGDW.

D.3.1 Setup and Update Equations

Let $w_t \in \mathcal{H}$ denote the function state at time t . Let ∇J_t denote the functional gradient of the loss at step t . The update dynamics are defined by the following system of recurrence relations:

$$\nu_t = \mu \nu_{t-1} + \nabla J_t, \quad (92)$$

$$w_t = (1 - \gamma)w_{t-1} - \eta \nu_t. \quad (93)$$

where:

- $\mu \in [0, 1)$ is the momentum coefficient.
- $\gamma \in [0, 1)$ is the weight decay factor per step.
- $\eta > 0$ is the learning rate.
- ν_t is the velocity function (initialized at $\nu_0 = 0$).
- w_0 is the initial function state.

D.3.2 Derivation of the Closed-Form State

We aim to express w_t as a direct summation of the history of gradients $\{\nabla J_j\}_{j=1}^t$. For convenience, we define the **effective momentum** $\tilde{\mu}$ and **effective learning rate** $\tilde{\eta}$ as:

$$\tilde{\mu} = \frac{\mu}{1 - \gamma}, \quad \tilde{\eta} = \frac{\eta}{1 - \gamma}.$$

First, we unroll the velocity term ν_t using equation 92:

$$\nu_k = \sum_{j=1}^k \mu^{k-j} \nabla J_j. \quad (94)$$

Next, we substitute this into the weight update equation 93. By unrolling the recurrence for w_t , we obtain:

$$\begin{aligned} w_t &= (1 - \gamma)w_{t-1} - \eta \nu_t \\ &= (1 - \gamma)^t w_0 - \eta \sum_{k=1}^t (1 - \gamma)^{t-k} \nu_k \\ &= (1 - \gamma)^t w_0 - \tilde{\eta} \sum_{k=1}^t (1 - \gamma)^{t-k+1} \nu_k. \end{aligned} \quad (95)$$

Substituting the expression for ν_k and rearranging the summation order ($1 \leq j \leq k \leq t$):

$$w_t = (1 - \gamma)^t w_0 - \tilde{\eta} \sum_{j=1}^t \nabla J_j \left(\sum_{k=j}^t (1 - \gamma)^{t-k+1} \mu^{k-j} \right). \quad (96)$$

We simplify the inner summation $\mathcal{C}(t, j)$ as follows:

$$\mathcal{C}(t, j) = \sum_{k=j}^t (1 - \gamma)^{t-k+1} \mu^{k-j} \quad (97)$$

$$\begin{aligned} &= \sum_{m=0}^{t-j} (1 - \gamma)^{(t-j+1)-m} \mu^m \\ &= (1 - \gamma)^{t-j+1} \sum_{m=0}^{t-j} \left(\frac{\mu}{1 - \gamma} \right)^m \\ &= (1 - \gamma)^{t-j+1} \sum_{m=0}^{t-j} \tilde{\mu}^m \end{aligned} \quad (98)$$

$$= (1 - \gamma)^{t-j+1} \frac{1 - \tilde{\mu}^{t-j+1}}{1 - \tilde{\mu}}. \quad (99)$$

Substituting this back into the expression for w_t , we arrive directly at the compact closed-form solution:

$$w_t = (1 - \gamma)^t w_0 - \tilde{\eta} \sum_{j=1}^t \left[(1 - \gamma)^{t-j+1} \frac{1 - \tilde{\mu}^{t-j+1}}{1 - \tilde{\mu}} \right] \nabla J_j. \quad (100)$$

Similarly to the implementation of momentum, we can store the creation time t_j for each tuple and then take it into account in the forward pass.

E Additional technical details on advanced linear attention layers

E.1 DeltaNet

In (unnormalized) linear attention, each step writes a new key-value association into a fixed size state via an outer-product update. The delta rule instead updates the value associated with the current key by (partly) overwriting the previously stored value with write strength $\lambda \in [0, 1]$, $\mathbf{v}_t^{\text{new}} = \lambda \mathbf{v}_t + (1 - \lambda) \mathbf{v}_t^{\text{old}}$ and $\mathbf{W}_t = \mathbf{W}_{t-1} - \mathbf{v}_t^{\text{old}} \mathbf{k}_t^\top + \mathbf{v}_t^{\text{new}} \mathbf{k}_t^\top$. In our setting, we set the key to be the activation, i.e., $\mathbf{k}_t := \mathbf{x}_t$ and define the previously stored value for that key as the current prediction $\mathbf{W}_{t-1} \mathbf{x}_t$. To compress a mapping from activations to (negative) gradients into a fixed size state as in vanilla gradient descent we set the target value to be the negative output-gradient scaled by a step size η , i.e., $\mathbf{v}_t := -\eta \nabla_{\mathbf{y}_t} \mathcal{L}$, which gives $\mathbf{v}_t^{\text{new}} = -\lambda \eta \nabla_{\mathbf{y}_t} \mathcal{L} + (1 - \lambda) \mathbf{W}_{t-1} \mathbf{x}_t$.

This yields the recurrent update

$$\mathbf{W}_t = \mathbf{W}_{t-1} (\mathbf{I} - \lambda \mathbf{x}_t \mathbf{x}_t^\top) + (-\lambda \eta \nabla_{\mathbf{y}_t} \mathcal{L}) \mathbf{x}_t^\top, \quad (101)$$

which shows that a DeltaNet-like update can be interpreted as gradient descent preceded by a targeted forgetting mechanism along the current activation direction. Notably, the update itself corresponds to performing a gradient descent step w.r.t. $\mathbf{W}$ on the layer-local, time-dependent loss

$$\ell_t(\mathbf{W}) := \frac{1}{2} \|\mathbf{W} \mathbf{k}_t - \mathbf{v}_t\|_2^2 = \frac{1}{2} \|\mathbf{W} \mathbf{x}_t + \eta \nabla_{\mathbf{y}_t} \mathcal{L}\|_2^2 \quad (102)$$

with learning rate λ [13, 19], meaning the Delta-rule-based update implicitly performs gradient descent within the gradient descent weight update. In practice, the update can be implemented with little computational overhead by rewriting Eq. 101 to

$$\mathbf{W}_t = \mathbf{W}_{t-1} + \lambda (-\eta \nabla_{\mathbf{y}_t} \mathcal{L} - \mathbf{W}_{t-1} \mathbf{x}_t) \mathbf{x}_t^\top, \quad (103)$$

caching the activation $\mathbf{W}_{t-1} \mathbf{x}_t$ from the forward pass and using it to form the corrected term $-\eta \nabla_{\mathbf{y}_t} \mathcal{L} - \mathbf{W}_{t-1} \mathbf{x}_t$ before computing the outer-product update.

E.2 MesaNet: Problem formulation and connection to FOOF

Next, we show the connection between FOOF and Mesa layer when applied to the database. We present a slightly modified formulation of the Mesa layer, with decay factor and an additional regularization center. We present the case of batch size 1, but the result can easily be extended to the general case.

Consider the online estimation of a parameter matrix $\mathbf{W} \in \mathbb{R}^{k \times d}$ from a stream of observations $(\mathbf{x}_t, \mathbf{y}_t)$ (key and value of the database) for $t = 1, \dots, T$. The mesa layer seeks to minimize the following time-discounted objective function $J_T(\mathbf{W})$ at each step T :

$$J_T(\mathbf{W}) = \sum_{t=1}^T \gamma^{T-t} \|\mathbf{y}_t - \mathbf{W} \mathbf{x}_t\|^2 + \Lambda_T \|\mathbf{W} - \mathbf{C}_T\|_F^2 \quad (104)$$

where:

- $\gamma \in [0, 1]$ is the forgetting factor.
- $\Lambda_T > 0$ is the regularization strength at time T .

- $\mathbf{C}_T \in \mathbb{R}^{k \times d}$ is the regularization center (prior mean) at time T , taken as the exponential moving average of past weights, defined as $\mathbf{C}_T = \gamma \frac{\Lambda_{T-1}}{\Lambda_T} \mathbf{C}_{T-1} + (1 - \gamma \frac{\Lambda_{T-1}}{\Lambda_T}) \mathbf{W}_{T-1}$, and $\mathbf{C}_0 = \mathbf{0}$.

The loss in the main text is recovered by choosing $\gamma = 1$ and $\Lambda_t = \lambda \forall t$.

It can be shown that the optimal solution $\mathbf{W}_T$ satisfies the normal equation:

$$\mathbf{W}_T \Phi_T = \mathbf{P}_T \quad (105)$$

where Φ_T and $\mathbf{P}_T$ are the regularized correlation matrix and cross-correlation vector, defined respectively as:

$$\Phi_T = \sum_{t=1}^T \gamma^{T-t} \mathbf{x}_t \mathbf{x}_t^\top + \Lambda_T \mathbf{I} \quad (106)$$

$$\mathbf{P}_T = \sum_{t=1}^T \gamma^{T-t} \mathbf{y}_t \mathbf{x}_t^\top + \Lambda_T \mathbf{C}_T \quad (107)$$

E.3 Derivation of the Recursive Update

We seek to express $\mathbf{W}_T$ as a function of $\mathbf{W}_{T-1}$ and the current observation $(\mathbf{x}_T, \mathbf{y}_T)$. First, we expand the recursive definitions of the sufficient statistics.

E.3.1 Recursive Matrix Update

Separating the term at $t = T$ and substituting Φ_{T-1} :

$$\begin{aligned} \Phi_T &= \gamma \left(\sum_{t=1}^{T-1} \gamma^{T-1-t} \mathbf{x}_t \mathbf{x}_t^\top \right) + \mathbf{x}_T \mathbf{x}_T^\top + \Lambda_T \mathbf{I} \\ &= \gamma (\Phi_{T-1} - \Lambda_{T-1} \mathbf{I}) + \mathbf{x}_T \mathbf{x}_T^\top + \Lambda_T \mathbf{I} \\ &= \gamma \Phi_{T-1} + \mathbf{x}_T \mathbf{x}_T^\top + (\Lambda_T - \gamma \Lambda_{T-1}) \mathbf{I} \end{aligned} \quad (108)$$

E.3.2 Recursive Vector Update

Similarly for $\mathbf{P}_T$:

$$\begin{aligned} \mathbf{P}_T &= \gamma (\mathbf{P}_{T-1} - \Lambda_{T-1} \mathbf{C}_{T-1}) + \mathbf{y}_T \mathbf{x}_T^\top + \Lambda_T \mathbf{C}_T \\ &= \gamma \mathbf{P}_{T-1} + \mathbf{y}_T \mathbf{x}_T^\top + (\Lambda_T \mathbf{C}_T - \gamma \Lambda_{T-1} \mathbf{C}_{T-1}) \end{aligned} \quad (109)$$

E.3.3 Sufficient Condition for the Delta Rule

Assuming $\mathbf{W}_{T-1} = \mathbf{P}_{T-1} \Phi_{T-1}^{-1}$, we substitute $\mathbf{P}_{T-1} = \mathbf{W}_{T-1} \Phi_{T-1}$ into the expression for $\mathbf{P}_T$:

$$\mathbf{P}_T = \mathbf{W}_{T-1} (\gamma \Phi_{T-1}) + \mathbf{y}_T \mathbf{x}_T^\top + (\Lambda_T \mathbf{C}_T - \gamma \Lambda_{T-1} \mathbf{C}_{T-1}) \quad (110)$$

From the matrix update step, we substitute $\gamma \Phi_{T-1} = \Phi_T - \mathbf{x}_T \mathbf{x}_T^\top - (\Lambda_T - \gamma \Lambda_{T-1}) \mathbf{I}$:

$$\mathbf{P}_T = \mathbf{W}_{T-1} (\Phi_T - \mathbf{x}_T \mathbf{x}_T^\top - (\Lambda_T - \gamma \Lambda_{T-1}) \mathbf{I}) + \mathbf{y}_T \mathbf{x}_T^\top + (\Lambda_T \mathbf{C}_T - \gamma \Lambda_{T-1} \mathbf{C}_{T-1}) \quad (111)$$

Rearranging terms to isolate the prediction error:

$$\mathbf{P}_T = \mathbf{W}_{T-1} \Phi_T + (\mathbf{y}_T - \mathbf{W}_{T-1} \mathbf{x}_T) \mathbf{x}_T^\top + \mathcal{R} \quad (112)$$

where the residual term $\mathcal{R}$ is defined as:

$$\mathcal{R} = (\Lambda_T \mathbf{C}_T - \gamma \Lambda_{T-1} \mathbf{C}_{T-1}) - \mathbf{W}_{T-1} (\Lambda_T - \gamma \Lambda_{T-1}) \quad (113)$$

Multiplying by Φ_T^{-1} from the right yields the update for $\mathbf{W}_T$:

$$\mathbf{W}_T = \mathbf{W}_{T-1} + (\mathbf{y}_T - \mathbf{W}_{T-1} \mathbf{x}_T) \mathbf{x}_T^\top \Phi_T^{-1} + \mathcal{R} \Phi_T^{-1} \quad (114)$$

For the residual term $\mathcal{R}$ to vanish, it is necessary and sufficient to impose the following consistency condition on the update of the regularization center $\mathbf{C}_T$:

$$\Lambda_T \mathbf{C}_T = \gamma \Lambda_{T-1} \mathbf{C}_{T-1} + (\Lambda_T - \gamma \Lambda_{T-1}) \mathbf{W}_{T-1}, \quad (115)$$

which is satisfied by definition.

Under the above condition, and using as target at every timestep $\mathbf{y}_T = \mathbf{W}_{T-1} \mathbf{x}_T - \eta \nabla_{\mathbf{y}_T} \mathcal{L}$, applying the loss in Equation 104 to the database would correspond to having a state, $\mathbf{W}, \Phi$, which is updated following:

$$\Phi_T = \gamma \Phi_{T-1} + \mathbf{x}_T \mathbf{x}_T^\top + (\Lambda_T - \gamma \Lambda_{T-1}) \mathbf{I} \quad (116)$$

$$\mathbf{W}_T = \mathbf{W}_{T-1} - \eta \nabla_{\mathbf{W}_{T-1}} \mathcal{L} \cdot \Phi_T^{-1} \quad (117)$$

It now only remains to define a meaningful Λ_T .

E.4 Proposed Algorithms

We identify two distinct regimes satisfying the consistency condition equation 115.

E.4.1 Algorithm A: Static Regularization

We assume a constant regularization strength $\Lambda_T = \lambda$ for all T .

Matrix Update:

$$\Phi_T = \gamma \Phi_{T-1} + \mathbf{x}_T \mathbf{x}_T^\top + \lambda(1 - \gamma) \mathbf{I}$$

Which corresponds exactly to the FOOF optimizer.

E.4.2 Algorithm B: Cumulative Regularization

The previous algorithm causes growing norm of Φ and a decrease in the importance of the prior, which may incur numerical and hyperparameter instabilities. We therefore propose the next algorithm, which we use in practice.

We assume the regularization strength accumulates over time according to $\Lambda_T = \lambda + \gamma \Lambda_{T-1}$. This is equivalent to optimizing the *weighted averaged* loss

$$J_T(\mathbf{W}) = \frac{1}{\sum_{t=1}^T \gamma^{T-t}} \sum_{t=1}^T \gamma^{T-t} \|\mathbf{y}_t - \mathbf{W} \mathbf{x}_t\|^2 + \lambda \|\mathbf{W} - \mathbf{C}_T\|_F^2 \quad (118)$$

Matrix Update:

$$\Phi_T = \gamma \Phi_{T-1} + \mathbf{x}_T \mathbf{x}_T^\top + \lambda \mathbf{I}$$

While keeping the above objective, we introduce additional tricks to stabilize numeric and hyperparameter across different layer widths, resulting in the following recursive update for a batch of B inputs $\mathbf{X}_T \in \mathbb{R}^{d \times B}$:

$$\Gamma_T = \gamma \Gamma_{T-1} + 1 \quad (119)$$

$$\Psi_T = \frac{\gamma \Gamma_{T-1}}{\Gamma_T} \Psi_{T-1} + \frac{1}{B d \Gamma_T} \mathbf{X}_T \mathbf{X}_T^\top \quad (120)$$

$$\mathbf{W}_T = \mathbf{W}_{T-1} - \eta \nabla_{\mathbf{W}_{T-1}} \mathcal{L} \cdot (\Psi_T + \lambda \mathbf{I})^{-1}, \quad (121)$$

while initializing $\Psi_0 = \mathbf{0}$, $\mathbf{W}_0$ to a standard weight initialization, and $\Gamma_0 = 0$.

F Experimental Setup

F.1 Tasks and Model Selection

We run experiments on image classification and teacher-student regression tasks. For all experiments, we report the mean and standard deviation over 5 seeds. To produce the results in Figure 2, we run each model/optimizer with a comparable set of hyperparameters where they are shared and for each method add the needed, additional hyperparameters. For Figures 1, 3, and 4, we select a subset of the hyperparameters relevant to the produced result and again detail the selection in the respective model subsection. The considered ranges of hyperparameters are reported beneath in the respective model subsections.

Generally, for each plot and for each method displayed in the respective plot, we select those hyperparameters that maximize/minimize the metric displayed in the plot. The only place where we share the selection across multiple plots is Fig. 2. There, the runs visualized in Fig. 2/Left and Fig. 2/Middle are tied in that the two metrics (Test Accuracy and Training Loss) are displayed for the same selection of runs that maximize the final test accuracy. In particular, looking for the hyperparameter that minimizes the training loss would yield a different selection of runs achieving even smaller training loss (but potentially worse generalization).

In the following two subsections, we first briefly describe the data processing/generation pipeline used for classification and regression experiments.

F.1.1 Image Classification

CIFAR-10 and CIFAR-100 [44] are image classification datasets on $32 \times 32 \times 3$ dimensional RGB images from 10 and 100 classes, respectively. Before handing these images to the network, we first apply standard normalization by subtracting the mean and dividing by the standard deviation per channel. Then, to increase the effective size of the training dataset, we apply standard image transformations, randomly sampled during training. These include reflect-padding (± 4 pixels) followed by a random 32×32 crop applied to all 3 channels and an epoch-wise horizontal flip. Test data is only normalized without applying augmentation. Before handing the images to our MLP network, we flatten them into a 3072-dimensional vector. The notion of epoch in the respective experiments refers to the following: take the 50000 training samples from the respective dataset, apply a randomly chosen augmentation (following the above rules) to each sample, and ultimately shuffle the samples. Only for Fig. 2/Right, where only a single pass on the training data is performed, we use the non-augmented standard version of CIFAR-10.

F.1.2 Teacher-Student Regression

We generate synthetic regression tasks using a random teacher network. The inputs $\mathbf{x} \in \mathbb{R}^{d_{\text{in}}}$ are sampled from a standard normal distribution and projected onto the unit sphere. The targets for the randomly sampled inputs are generated by a randomly initialized MLP teacher with L residual hidden blocks adding to a residual stream of width d_e . Crucially, this teacher has nonlinearities thereby, upon initialization, implementing some nonlinear input-output(target) mapping. Particularly, the teacher first projects the input data to the hidden dimension by applying a linear layer and followed by a ReLU activation. Subsequently, $L - 1$ residual blocks, where each block consists of a linear layer and a ReLU activation, are applied to the initial residual stream activation. A final linear projection maps the last residual stream activation mapping to the output feature (the target for the regression task) of dimension d_{out} . All weights are randomly initialized (standard variance scaling init). In our experiments, we set $d_{\text{in}} = 100$, $d_e = 100$, $L = 2$, and $d_{\text{out}} = 1$.

F.2 Model Architecture

Throughout the entire paper, and importantly shared across all methods, all experiments (unless explicitly stated otherwise) are run with a ResNet-MLP architecture. The inputs act as the initial residual stream activation, thereby governing the residual stream dimension $d_e = d_{\text{in}}$. The network backbone consists of L residual blocks arranged in a pre-normalization configuration. Specifically, each block applies layer normalization, followed by a trainable module (a standard dense layer or one of the proposed alternatives: Delta, Mesa, RBF, or Weighted Softmax). In the case of a standard dense layer, Delta, or Mesa, a non-linear activation function is applied to the output of the block and

Table 2: Hyperparameters Nonparametric Growing Weighted Softmax for Teacher-Student experiments.

Hyperparameter	Value
Model	Single Weighted Softmax layer
Data normalization	RMSNorm (on input)
Value learning rate β	[0.1, 0.3, 1.0]
Weight learning rate η	[10.0, 30.0, 100.0]
Softmax temperature T	[0.1, 1.0, 3.0, 10.0]
Momentum	[0.6, 0.9]
Batch size	64
Epochs	200000

the norm rescaling parameters are trained. Note that for RBF and Weighted Softmax, nonlinearities are by design included in the block. Also, for those we do not train the scale parameters of the norm but fix them to be unit. In all cases, the block’s output is added back to the residual stream. The final residual stream activation is processed by a final normalization (again scale parameters not trained for RBF and Softmax and trained for all others) before being mapped to the C class logits via a final output block (again, either a dense layer or one of the proposed methods). In the following subsections, we detail the various instantiations of the residual blocks with the methods described in this work.

F.3 Weighted Softmax

We evaluate Weighted Softmax kernel-based RCDL models on both synthetic teacher-student regression and image classification tasks, using the classic CIFAR-10 and CIFAR-100 benchmarks. For these experiments, we instantiate the blocks of the ResNet-MLP backbone architecture, described in Appendix F.2, with Weighted Softmax RCDL modules.

Algorithm 3 Growing Weighted Softmax Attention

- 1: **Input:** Stream of query batches $\{\mathbf{q}_{t,n}\}_{n=1}^B$, temperature T , value-learning rate $\beta > 0$, weight-learning rate $\eta > 0$, $\mathbf{W}_0 = \frac{1}{B}$, $\mathbf{K}_0 \sim \mathcal{N}(0, \frac{1}{\sqrt{d_{\text{in}}}} \mathbf{I})$, $\mathbf{V}_0 \sim \mathcal{N}(0, \frac{1}{\sqrt{d_{\text{out}}}} \mathbf{I})$
 - 2: **Initialize:** $\mathcal{D}_0 \leftarrow \{(\mathbf{W}_0, \mathbf{K}_0, \mathbf{V}_0)\}$
 - 3: **for** $t = 1, 2, \dots$ **do**
 - 4: **for** $n = 1, \dots, B$ **do**
 - 5: Compute prediction: $\mathbf{s}_{t,n} \leftarrow \frac{\sum_{(w, \mathbf{k}, \tilde{\mathbf{v}}) \in \mathcal{D}_{t-1}} e^{\frac{1}{T\sqrt{d_{\text{in}}}} \langle \mathbf{k}, \mathbf{q}_{t,n} \rangle} \tilde{\mathbf{v}}}{\sum_{(w, \mathbf{k}, \tilde{\mathbf{v}}) \in \mathcal{D}_{t-1}} w e^{\frac{1}{T\sqrt{d_{\text{in}}}} \langle \mathbf{k}, \mathbf{q}_{t,n} \rangle}}$
 - 6: Compute gradient: $\mathbf{g}_{t,n} \leftarrow \nabla \mathcal{L}(\mathbf{s}_{t,n})$
 - 7: Compute inner product: $\delta_{t,n} \leftarrow \langle \mathbf{s}_{t,n}, \mathbf{g}_{t,n} \rangle$
 - 8: **end for**
 - 9: **Update database:** $\mathcal{D}_t \leftarrow \mathcal{D}_{t-1} \cup \{(\max\{0, \eta \delta_{t,n}\}, \mathbf{q}_{t,n}, -\beta \mathbf{g}_{t,n}) : n = 1, \dots, B\}$
 - 10: **end for**
-

A Weighted Softmax module maintains a growing database of key, value, weight tuples $\mathcal{D}_t = (\mathbf{K}_t, \mathbf{V}_t, \mathbf{W}_t) = (\{\mathbf{k}_i, \mathbf{v}_i, w_i\}_{i=0}^t)$ of size $(t+1)B$ at time step t , where B is the batch size. For numerical stability, we initialize this database with a randomly sampled batch at $t = 0$. Particularly, we set $\mathcal{D}_0 = (\mathbf{K}_0, \mathbf{V}_0, \mathbf{W}_0)$. Here, $\mathbf{K}_0 \in \mathbb{R}^{B \times d_e}$ and $\mathbf{V}_0 \in \mathbb{R}^{B \times d_{\text{out}}}$ ($d_{\text{out}} = d_e$ for the residual blocks) are sampled from a standard normal distribution and subsequently normalized along the feature dimension by root mean squared normalization. All entries of $\mathbf{W}_0 \in \mathbb{R}^B$ are set to a constant weight $w_0 = \frac{1}{B}$, where B is the batch size. At timestep $t > 0$, the database is grown following the steps outlined in Algorithm 3. Briefly, we apply the Weighted Softmax update rule but clip negative weights to 0 (cf. Appendix C.5). Moreover, for numerical stability, we drop the division by the normalization D_{t-1} for both the weight and the value update. Ultimately, we introduce learning rates η and β to scale weights and values independently.

Table 3: Hyperparameters Growing Weighted Softmax for experiments on CIFAR-10 and CIFAR-100.

Hyperparameter	Value
Dataset	CIFAR-10, CIFAR-100
Input	$32 \times 32 \times 3$ images flattened to 3072
Model	Residual MLP
Residual stream size (d_e)	3072
# residual blocks (L)	[0, 1, 2]
Normalization in blocks	RMSNorm (pre-norm, per block)
Final normalization	RMSNorm before head
Classifier head	Weighted softmax-kernel RCDL layer
Weight learning rate η	[0.3, 1.0, 1.5, 3.0]
Value learning rate β	[20.0, 30.0, 40.0, 60.0]
Softmax temperature T	[1.0, 3.0, 10.0, 15.0, 30.0]
Momentum	[0.0, 0.1, 0.9, 0.999]
Batch size	64
Epochs	50

We report the hyperparameters considered to produce the Weighted Softmax numbers in Figure 1 in Table 2. Additionally, the dimensionality of the inputs and thus of the keys in the synthetic regression task is $d_e = 100$, Input = 100.

The hyperparameters considered to produce the Weighted Softmax numbers in Figure 2 and 3 are reported in Table 3. Both Figures display data from the same set of experiments. On top, for Figure 2/Right we also include the batch size in the sweeps and consider the values 4, 16, 32 and 64. For the Weighted Softmax numbers reported in Figure 4 we again follow Table 3 but only consider models with $L = 0$ residual blocks.

F.3.1 Gauge-Invariant Weighted Softmax

The Gauge-Invariant Weighted Softmax module uses the same database, initialization and prediction as the Weighted Softmax module (Appendix F.3). Only the weight and value of each new tuple differ (Algorithm 2). As explained in Appendix C.7, we can choose the gauge offset b_t so that the added weight $w_t = \eta\alpha\|g_t\| \geq 0$ is non-negative by construction, so the clipping $\max\{0, \cdot\}$ of Algorithm 3 is no longer needed. As before, we drop the division by D_{t-1} in both updates. In practice we add $\epsilon = 10^{-8}$ to $\|g_t\|$ in the denominator for numerical stability.

F.4 Hybrid Weighted Softmax

The Hybrid Weighted Softmax layer keeps the database $\mathcal{D}$ of the Weighted Softmax (Appendix F.3), but differs in two ways (Algorithm 4). First, all stored tuples are treated as parameters and trained by backpropagation with standard gradient descent optimization. Second, a query is only added if its Euclidean distance to the nearest stored key exceeds the growth threshold τ . The new tuple follows the Weighted Softmax rule of Algorithm 3, with $\delta_{t,n}$ clipped to $[0, \infty)$ for stability. $\mathcal{D}_0$ consists of B tuples with zero keys and values and weight $\frac{1}{B}$. Because queries are RMS-normalized, $\|q\|_2 = \sqrt{d_{\text{in}}} \approx 55.4$, which sets the scale of τ . We define capacity as the total number of stored tuples after training, summed over all layers and divided by the maximum $4N$.

Table 4 lists the hyperparameters for Figure 5.

F.5 Parametric Softmax

Here we describe the Parametric Softmax baseline that is referenced in Figures 1 and 4. The motivation for this baseline is to compare growing a (weighted) softmax kernel-based module with learning a fixed size key-value parameterization for a softmax kernel where all keys and values are treated as parameters i.e., are optimized jointly to minimize the loss.

Algorithm 4 Hybrid Weighted Softmax Attention

- 1: **Input:** Stream of query batches $\{q_{t,n}\}_{n=1}^B$, temperature T , growth threshold τ , maximal size N , value-learning rate $\beta > 0$, weight-learning rate $\eta > 0$, Adam learning rate α , $W_0 = \frac{1}{B}$, $K_0 \sim \mathcal{N}(0, \frac{1}{\sqrt{d_{\text{in}}}}I)$, $V_0 \sim \mathcal{N}(0, \frac{1}{\sqrt{d_{\text{out}}}}I)$
 - 2: **Initialize:** $\mathcal{D}_0 \leftarrow \{(W_0, K_0, V_0)\}$
 - 3: **for** $t = 1, 2, \dots$ **do**
 - 4: **for** $n = 1, \dots, B$ **do**
 - 5: Compute prediction: $s_{t,n} \leftarrow \frac{\sum_{(w, k, \tilde{v}) \in \mathcal{D}_{t-1}} e^{\frac{1}{T\sqrt{d_{\text{in}}}} \langle k, q_{t,n} \rangle} \tilde{v}}{\sum_{(w, k, \tilde{v}) \in \mathcal{D}_{t-1}} w e^{\frac{1}{T\sqrt{d_{\text{in}}}} \langle k, q_{t,n} \rangle}}$
 - 6: Compute gradient: $g_{t,n} \leftarrow \nabla \mathcal{L}(s_{t,n})$
 - 7: Compute inner product: $\delta_{t,n} \leftarrow \langle s_{t,n}, g_{t,n} \rangle$
 - 8: **end for**
 - 9: **Select novel queries:** $\mathcal{S}_t \leftarrow \{n : \min_{(\cdot, k, \tilde{v}) \in \mathcal{D}_{t-1}} \|q_{t,n} - k\|_2 > \tau\}$, keeping only the first $N - |\mathcal{D}_{t-1}|$ elements if necessary
 - 10: **Parametric update:** update all $(w, k, \tilde{v}) \in \mathcal{D}_{t-1}$ with one Adam step (learning rate α) on $\frac{1}{B} \sum_n \mathcal{L}(s_{t,n})$
 - 11: **Update database:** $\mathcal{D}_t \leftarrow \mathcal{D}_{t-1} \cup \{(\max\{0, \eta \delta_{t,n}\}, q_{t,n}, -\beta g_{t,n}) : n \in \mathcal{S}_t\}$
 - 12: **end for**
-

Table 4: Hyperparameters for the Hybrid Growing Weighted Softmax on CIFAR-10 (Figure 5).

Hyperparameter	Value
Model	$L = 3$ residual blocks + readout layer, no nonlinearity
Embedding dimension d_e	3072 (flattened image, no input embedding)
Normalization	RMSNorm (no learned scale) before each layer and the readout
Maximal width per layer N	50016
Growth threshold τ	[35, 40, 45, 50, 55, 60]
Value learning rate β	[10, 30, 60]
Weight learning rate η	1.0
Softmax temperature T	[1.0, 5.0, 10.0]
Adam learning rate α	[0.0003, 0.001, 0.003]
Batch size	16
Epochs	1

Unlike the growing module, this baseline uses a static architecture with fixed size parameters (keys, values and weights), pre-allocated with enough capacity to match the size of a fully grown database from the RCDL case from the beginning. We obtain predictions s_{t-1} applying a softmax to this huge database, but instead of growing the database step-wise, update the entire key-value parameterization by gradient descent at every timestep. Essentially, this baseline can be seen as wide MLP with a softmax nonlinearity.

We report the hyperparameters considered to produce Parametric Softmax numbers in CIFAR-10 experiments (cf. Figure 4) in Table 5.

For the synthetic Teacher-Student regression task presented in Figure 1, we adapt the "database size" of our fixed parametric representations to shapes $(10240000, d_{\text{in}})$ for W_{key} and $(10240000, d_{\text{out}})$ for W_{value} . We report the remaining hyperparameters for this result in Table 6. We find that training the weights for a weighted version of softmax leads to severe training instabilities in this parametric setup and hence report the numbers for an unweighted softmax variant.

F.6 RBF

We evaluate RBF kernel-based RCDL models on image classification tasks, using the classic CIFAR-10 and CIFAR-100 benchmarks. For these experiments, we instantiate the blocks of the ResNet-MLP backbone architecture, described in Appendix F.2, with RBF RCDL modules. In all RBF kernel-based RCDL modules, we predict and grow the database according to Algorithm 5.

Table 5: Hyperparameters Parametric Softmax for experiments on CIFAR-10.

Hyperparameter	Value
Dataset	CIFAR-10
Input	$32 \times 32 \times 3$ images flattened to 3072
Model	MLP
Normalization in blocks	RMSNorm (pre-norm, per block)
W_{Key} shape	(2560000, 3072)
W_{Value} shape	(2560000, 10)
Nonlinear Activation	Softmax
Learning rate α	[0.003, 0.01, 0.03, 0.1]
Softmax temperature T	[0.001, 0.03, 0.1, 0.3, 1.0, 3.0, 10.0]
Batch size	64
Epochs	2000

Table 6: Hyperparameters Parametric Softmax for Teacher-Student experiments.

Hyperparameter	Value
Model	Single Parametric Softmax layer
Hidden size	10240000
Data normalization	RMSNorm (on input)
Learning rate η	[0.0003, 0.001, 0.003, 0.01, 0.03]
Temperature T	[0.001, 0.01, 0.1, 0.3, 1.0, 3.0, 10.0, 30.0, 100.0]
Optimizer	Adam
Momentum	0.85
Batch size	64
Training steps	200000

We report the hyperparameters considered to produce the RBF numbers in Figure 2 in Table 7. On top, for Figure 2/Right we also include the batch size in the sweeps and consider the values 4, 16, 32 and 64.

F.7 Mesa (FOOF)

We evaluate the performance of the Mesa update rule introduced in Section 4 on CIFAR-10 and CIFAR-100 benchmarks. For these experiments, we instantiate the blocks of the ResNet-MLP backbone architecture, described in Appendix F.2, with vanilla dense linear layers. We train the parameters of the resulting network by applying the Mesa weight update detailed in Section 4.

We report the hyperparameters used to produce the numbers for the Mesa update rule in Figure 2 in Table 9. We implement the computation of the matrix inverse (cf. Appendix E) using the Newton-Schulz Iteration. We report the effect of iterations for this algorithm in Table 8.

Algorithm 5 Growing RBF

- 1: **Input:** Stream of query batches $\{\mathbf{q}_{t,n}\}_{n=1}^B$, temperature T , value-learning rate $\eta > 0$
 - 2: **Initialize:** $\mathcal{D}_0 \leftarrow \emptyset$
 - 3: **for** $t = 1, 2, \dots$ **do**
 - 4: **for** $n = 1, \dots, B$ **do**
 - 5: Compute prediction: $\mathbf{s}_{t,n} \leftarrow \sum_{(\mathbf{k}, \mathbf{v}) \in \mathcal{D}_{t-1}} \exp\left(-\frac{\|\mathbf{q}_{t,n} - \mathbf{k}\|_2^2}{2T\sqrt{d_{\text{in}}}}\right) \mathbf{v}$
 - 6: Compute gradient: $\mathbf{g}_{t,n} \leftarrow \nabla \mathcal{L}(\mathbf{s}_{t,n})$
 - 7: **end for**
 - 8: **Update database:** $\mathcal{D}_t \leftarrow \mathcal{D}_{t-1} \cup \{(\mathbf{q}_{t,n}, -\eta \mathbf{g}_{t,n}) : n = 1, \dots, B\}$
 - 9: **end for**
-

Table 7: Hyperparameters RBF for experiments on CIFAR-10 and CIFAR-100.

Hyperparameter	Value
Dataset	CIFAR-10, CIFAR-100
Input	$32 \times 32 \times 3$ images flattened to 3072
Model	Residual MLP
Residual stream size (d_e)	3072
# residual blocks (L)	$[0, 1, 2]$
Normalization in blocks	RMSNorm (pre-norm, per block)
Final normalization	RMSNorm before head
Classifier head	Weighted RBF-kernel RCDL layer
Value learning rate η	$[0.3, 1.0, 3.0]$
Temperature T	$[0.5, 1.0, 1.5, 5.0, 15.0]$
Momentum	$[0.0, 0.1, 0.9, 0.999]$
Batch size	64
Epochs	50

Table 8: Best Average Accuracy on CIFAR-10 and CIFAR-100 across different numbers of steps for the newton iteration algorithm for approximating the inverse. We find that generalization performance on CIFAR saturates with 10 iterations.

Dataset	Mesa newton-approximation steps		
	5	10	Inf.
CIFAR-10	0.6914	0.7130	0.7124
CIFAR-100	0.4149	0.4323	0.4319

F.8 Delta

Lastly, we evaluate the performance of the Delta rule variant introduced in Section 4 on the CIFAR-10 and CIFAR-100 benchmark. For these experiments, we instantiate the blocks of the ResNet-MLP backbone architecture, described in Appendix F.2, with vanilla dense linear layers. We train the parameters of the resulting network by applying the Delta weight update detailed in Section 4.

We report the hyperparameters used to produce the numbers for the Delta update rule in Figure 2 in Table 10. On top, for Figure 2/Right we also include the batch size in the sweeps and consider the values 4, 16, 32 and 64.

F.8.1 Additional Experimental Results: Extended training for deep networks with Delta regularization

Here, we present additional experimental results to evaluate the DeltaNet-like gradient descent update, which were obtained in a different experimental setting compared to the results in Section 5, to better assess the regularizing effect for deep neural networks. Specifically, we train a 10-layer ResNet MLP on Cifar-10 for 2000 epochs. Our architecture downprojects the preprocessed and flattened input to a hidden size of 512, and uses 2-layer MLPs with 4x expansion ratio, GELU activation [56] and a pre-block Layer norm [57]. We train with a batch size of 64 and implement the DeltaNet update in the batch setting by applying all decays within a batch simultaneously, that is, multiplying the linear layer by $\mathbf{I} - \lambda \mathbf{X}_t \mathbf{X}_t^\top$ before applying the gradient update, where $\mathbf{X}_t \in \mathbb{R}^{d_{in} \times B}$ denotes the stacked input activations from batch t . In contrast to the derivation in Section 4, we reparametrize the update rule to align the decay rate with the learning rate scheduling (analogous to standard weight decay). Specifically, we update the weights as

$$\mathbf{W}_t = \mathbf{W}_{t-1}(\mathbf{I} - \lambda \eta_t \mathbf{X}_t \mathbf{X}_t^\top) - \eta_t \mathbf{G}_{\mathbf{Y}_t} \mathbf{X}_t^\top, \quad (122)$$

where $\mathbf{G}_{\mathbf{Y}_t}$ contains the stacked output gradients, λ is the Delta rule decay rate and η_t is the learning rate, for which we use cosine annealing with linear warmup. A more detailed description of the experimental setup can be found in Table 11.

Table 9: Hyperparameters Mesa for experiments on CIFAR-10 and CIFAR-100.

Hyperparameter	Value
Dataset	CIFAR-10, CIFAR-100
Input	$32 \times 32 \times 3$ images flattened to 3072
Model	Residual MLP
Residual stream size (d_e)	3072
# residual blocks (L)	[2, 4]
Normalization in blocks	RMSNorm (pre-norm, per block)
Final normalization	RMSNorm before head
MLP activation	GELU
Classifier head	Linear ($3072 \rightarrow 10$)
Mesa- γ	[0.9, 0.99, 1.0]
Mesa- λ	[0.001, 0.01, 0.1]
Optimizer	Adam
Learning rate α	[0.0001, 0.0003, 0.001, 0.003, 0.01]
Momentum	0.85
Batch size	64
Epochs	50

Table 10: Hyperparameters Delta for experiments on CIFAR-10 and CIFAR-100.

Hyperparameter	Value
Dataset	CIFAR-10, CIFAR-100
Input	$32 \times 32 \times 3$ images flattened to 3072
Model	Residual MLP
Residual stream size (d_e)	3072
# residual (L)	[2, 4]
Normalization in blocks	RMSNorm (pre-norm, per block)
Final normalization	RMSNorm before head
MLP activation	GELU
Decay rate λ	[0.0000003, 0.000001, 0.000003, 0.00001, 0.0001, 0.0003]
Classifier head	Linear ($3072 \rightarrow 10$)
Learning rate α	[0.0001, 0.0003, 0.001, 0.003, 0.01]
Batch size	64
Epochs	50

We compare plain SGD, SGD with weight decay, and SGD with Delta rule decay when training for 2000 epochs. Note that this is significantly longer than in other experiments (with a significantly deeper model), as it is computationally feasible and allows us to better understand the effect of regularization. As shown in Fig. 6, adding a DeltaNet-like decay mechanism significantly improves the final test accuracy (74% vs. 67%), far exceeding that of multiplicative weight decay (Fig. 7). While both mechanisms lead to a much slower increase in train accuracy, only the DeltaNet-like decay achieves superior generalization. We also tested combining both decay mechanisms (as in Gated DeltaNet), but this yielded no additional improvement.

Additionally, we experimented with normalized DeltaNet updates as is standard for sequence modeling, i.e., using an input-dependent write strength of $\lambda\eta/\|x\|_2^2$. However, this slightly degraded performance. A likely explanation is that faithfully following the DeltaNet derivation would require scaling the learning rate accordingly; doing so, however, leads to highly unstable optimization, since it makes the (effective) per-sample learning rate dependent on the activation norm.

F.9 MLP Baselines

We complement the RCDL modules discussed in this work with strong MLP baselines on image classification tasks, using the classic CIFAR-10 and CIFAR-100 benchmarks. For these experiments, we instantiate the blocks of the ResNet-MLP backbone architecture, described in Appendix F.2, with

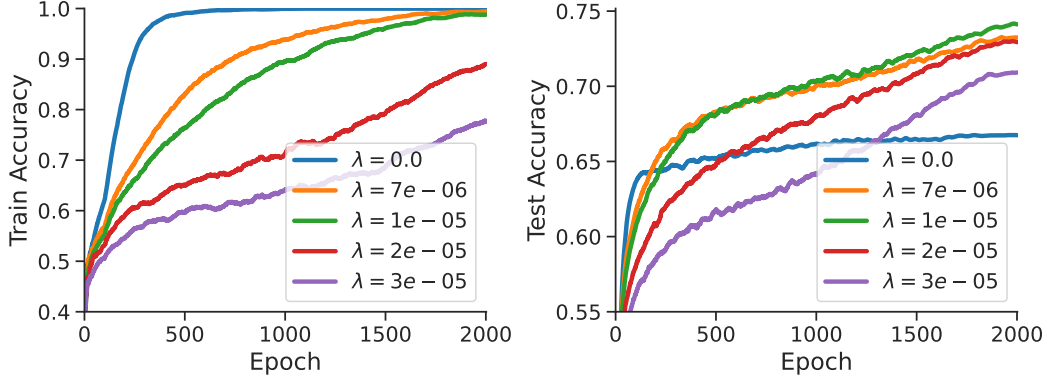


Figure 6: Training and test accuracy for a 10-layer residual MLP trained with SGD under varying DeltaNet decay rate λ . While $\lambda = 0$ achieves near-perfect training accuracy quickly, test accuracy plateaus at around 67%. Modest DeltaNet-like decay rates significantly improve generalization, as indicated by the reduced train-test gap, which demonstrates the regularizing effect of the update rule. Curves are smoothed using a running average.

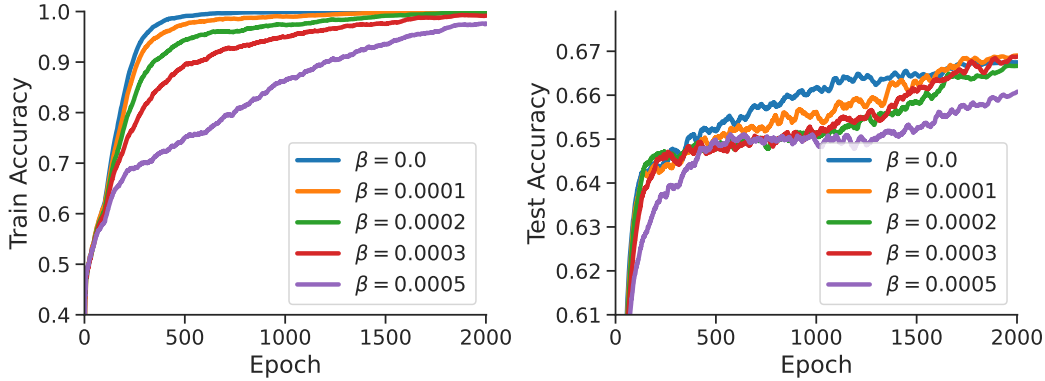


Figure 7: Training and test accuracy for a 10-layer residual MLP trained with SGD under varying multiplicative decay rate β . Similarly to DeltaNet-like decay the training accuracy increases much slower, however, generalization improves only marginally. Curves are smoothed using a running average.

vanilla dense linear layers. We train the parameters of the resulting network with either SGD [58], RMSProp [40], Adam [38], AdamW [39], and Muon [27].

We report the hyperparameters used to produce the numbers for the respective optimizer in Figure 2 in Table 12. On top, for Figure 2/Right we also include the batch size in the sweeps and consider the values 4, 16, 32 and 64.

We report the hyperparameters used to produce the numbers for the reference MLP in the Teacher-Student setup in Figure 1 in Table 13.

F.10 Nonparametric Baselines

In Figure 2/Right we complement the RCDL methods with two nonparametric baselines: A kernel ridge regression (least squares (LS-)SVM) on the 10-dimensional one-hot targets and a k-Nearest Neighbour (kNN) implementation.

Table 11: Hyperparameters used for DeltaNet-GD experiments on extended training on Cifar-10 with deep nets.

Hyperparameter	Value
Dataset	CIFAR-10
Input	$32 \times 32 \times 3$ images flattened to 3072
Model	Residual MLP
Input projection	Linear ($3072 \rightarrow 512$)
Residual stream size (d_e)	512
# residual blocks	10
Normalization in blocks	LayerNorm (pre-norm, per block)
Final normalization	LayerNorm before head
MLP expansion	$4\times$
MLP hidden dimension	2048
MLP activation	GELU
Dropout	0.1
Classifier head	Linear ($512 \rightarrow 10$)
Optimizer	SGD
Learning rate α	1×10^{-2}
Momentum	0.0
LR schedule	Step-wise: 5 epoch linear warmup + cosine annealing
Cosine schedule	CosineAnnealingLR over remaining steps
Batch size	64
Epochs	2000

Table 12: Hyperparameters MLP Baselines for experiments on CIFAR-10 and CIFAR-100.

Hyperparameter	Value
Dataset	CIFAR-10, CIFAR-100
Input	$32 \times 32 \times 3$ images flattened to 3072
Model	Residual MLP
Residual stream size (d_e)	3072
# residual (L)	[2, 4]
Normalization in blocks	RMSNorm (pre-norm, per block)
Final normalization	RMSNorm before head
MLP activation	GELU
Classifier head	Linear ($3072 \rightarrow 10$)
Optimizer	[SGD, RMSProp, Adam, AdamW, Muon]
Learning rate α	[0.0001, 0.0003, 0.001, 0.003, 0.01]
Weight decay (for AdamW)	[0.0001, 0.01]
Momentum	0.0
Batch size	[16, 64]
Epochs	50

F.10.1 Kernel ridge regression (least squares (LS-)SVM)

For the LS-SVM we chose the RBF-kernel $k(\mathbf{x}, \mathbf{x}') = \exp(-\gamma \|\mathbf{x} - \mathbf{x}'\|_2^2)$ inducing the Gram matrix $\mathbf{K} \in \mathbb{R}^{n \times n}$, with $K_{i,j} = k(\mathbf{x}_i, \mathbf{x}_j)$, where $\mathbf{x}_i, \mathbf{x}_j$ are the channel-wise normalized RGB images, flattened into 3072 dimensional vectors. We parametrize the bandwidth parameter following the median heuristic $\gamma = \frac{\gamma_{\text{mult}}}{\text{median}_{i,j} \|\mathbf{x}_i - \mathbf{x}_j\|_2^2}$ and approximate the sample median on all pairwise distances of a subsample of 2048 images. With the one-hot targets $\mathbf{Y} \in \mathbb{R}^{n \times 10}$ we invoke a standard solver to compute $\boldsymbol{\alpha} = (\mathbf{K} + \lambda \mathbf{I}_n)^{-1} \mathbf{Y} \in \mathbb{R}^{n \times 10}$ and evaluate $f(\mathbf{x}) = \sum_{i=1}^n \alpha_i k(\mathbf{x}, \mathbf{x}_i) \in \mathbb{R}^{10}$ on the test samples $\mathbf{x}$ (predicting the argmax class).

We report the considered hyperparameter ranges for producing the data-point displayed in Figure 2/Right in Table 14.

Table 13: Hyperparameters MLP Baseline for Teacher-Student setup.

Hyperparameter	Value
Model	Residual MLP
Residual stream size (d_e)	[256, 512, 1024, 4096, 8192]
# residual (L)	2
Normalization	None
MLP activation	ReLU
Optimizer	SGD
Learning rate	[0.0003, 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1.0, 3.0]
Momentum	[0.0, 0.9, 0.999]
Batch size	[64, 256, 1024]
Training steps	200000

Table 14: Hyperparameters RBF kernel regression for experiments on CIFAR-10.

Hyperparameter	Value
Dataset	CIFAR-10
Input	$32 \times 32 \times 3$ images flattened to 3072
Model	RBF kernel
Classification	arg max on predicted output
Optimizer	Exact solver
Bandwidth γ_{mult}	[0.25, 0.5, 1.0, 2.0, 3.0, 5.0]
Ridge regulariser λ	[0.001, 0.01, 0.1, 0.3, 1.0]

F.10.2 k-Nearest Neighbors (kNN)

We implement a kNN search using similarity score $s(\mathbf{x}, \mathbf{x}_i)$ on the channel-wise normalized and flattened test image and respective database entry $\mathbf{x}, \mathbf{x}_i \in \mathbb{R}^{3072}$. We consider two similarity scores

$$s(\mathbf{x}, \mathbf{x}_i) = \begin{cases} -\|\mathbf{x} - \mathbf{x}_i\|_2^2 & \text{euclidean} \\ \mathbf{x}^\top \mathbf{x}_i & \text{unnormalised dot product.} \end{cases}$$

Denoting $\mathcal{N}_k(\mathbf{x})$ as the indices of the top-k neighbors (according to the similarity score) we then weight the labels of these entries either uniformly or proportional to the respective similarity score leading to one of the following weights

$$w_i = \begin{cases} 1 & \text{uniform (regardless of similarity score)} \\ \frac{1}{\|\mathbf{x} - \mathbf{x}_i\|_2 + 10^{-5}} & \text{proportional, euclidean (inverse distance)} \\ \frac{\exp(\mathbf{x}^\top \mathbf{x}_i)}{\sum_{j \in \mathcal{N}_k(\mathbf{x})} \exp(\mathbf{x}^\top \mathbf{x}_j)} & \text{proportional, dot (softmax over top-k),} \end{cases}$$

for $i \in \mathcal{N}_k(\mathbf{x})$. Using these weights, the prediction for an input $\mathbf{x}$ is given by taking the argmax over

$$\frac{\sum_{i \in \mathcal{N}_k(\mathbf{x})} w_i 1[y_i = c]}{\sum_{i \in \mathcal{N}_k(\mathbf{x})} w_i}.$$

To obtain the number reported in Figure 2/Right we sweep the grid of the two similarity metrics, the two weighting mechanisms, and $k \in \{5, 10, 15, 20, 25\}$.

F.11 Vision Transformer Experiments

We investigate the performance of RCDL modules beyond ResNet architectures in classic vision Transformers (ViTs). We employ pre-LayerNorm transformer blocks consisting of standard softmax-attention and feed-forward networks (FFNs) and a linear readout on the CLS token. In the Weighted Softmax ViT, we replace each block’s FFN with a single Growing Weighted Softmax Module (Algorithm 3), all other components of the model are trained with standard parametric learning. Both are trained for a single epoch on CIFAR-10 (Appendix F.1.1), with hyperparameters in Tables 15 and 16.

Table 15: Hyperparameters Standard ViT for single-epoch experiments on CIFAR-10.

Hyperparameter	Value
Dataset	CIFAR-10
Input	$32 \times 32 \times 3$ images, 4×4 patches (64 patch tokens + CLS token)
Model	ViT, 2 transformer blocks
Embedding dimension d_e	128
Attention heads	8
Positional embedding	learned
Normalization in blocks	LayerNorm (pre-norm, per block) + final LayerNorm
FFN	Dense (128 $\rightarrow$ 512) $\rightarrow$ GELU $\rightarrow$ Dense (512 $\rightarrow$ 128)
FFN blowup factor	[1, 2, 4, 8, 16, 32]
Readout	Linear layer on CLS token
Optimizer	Adam
Learning rate α	[0.0003, 0.001, 0.003]
Batch size	64
Epochs	1
Seeds	5

Table 16: Hyperparameters Weighted Softmax ViT for single-epoch experiments on CIFAR-10.

Hyperparameter	Value
Dataset	CIFAR-10
Input	$32 \times 32 \times 3$ images, 4×4 patches (64 patch tokens + CLS token)
Model	ViT, 2 transformer blocks, FFN replaced by growing GWS
Embedding dimension d_e	128
Attention heads	8
Positional embedding	learned
Normalization in blocks	LayerNorm (pre-norm, per block) + final LayerNorm
Readout	Linear layer on CLS token
Optimizer (non-growing params.)	Adam
Learning rate (Adam) α	[0.0003, 0.001, 0.003]
Weight learning rate η	[0.1, 0.3, 1.0]
Value learning rate β	[10, 30]
Softmax temperature T	[0.1, 0.3, 1.0, 3.0]
Value momentum	[0.0, 0.999]
Batch size	64
Epochs	1
Seeds	5

F.12 Computational considerations

Growing Weighted Softmax, RBF-kernel based RCDL, and Parametric Weighted Softmax modules have a significant memory footprint. In the reported CIFAR- $\{10, 100\}$ experiments, where each epoch consists of 5×10^4 training examples, we add 2.5×10^6 tuples (w, v, k) (or (v, k) , respectively, for RBF-kernel based modules) to $\mathcal{D}$ in growing modules. For a hidden layer with $d_{\text{in}} = d_{\text{out}} = 3072$, implemented in float32, as used in CIFAR experiments, this results in a total memory footprint of 61.5GB (sum of memory footprint for both keys, values and weights). To realize the training we rely on parallelization and shard this database across 4 TPUs for 50-epoch CIFAR experiments. For the teacher-student regression experiments, training for 200 epochs adds 10^7 tuples to $\mathcal{D}$ in growing modules. For a hidden layer with $d_{\text{in}} = d_{\text{out}} = 100$, this results in a minimal memory footprint of approximately 8GB. As a result, this task does not require parallelization or database sharding and can be efficiently trained on a single device.

However, note that the update is relatively cheap in growing as we do not have to compute gradients for these large databases. This is in contrast to Parametric Weighted Softmax modules, where we furthermore have to consider the additional footprint of the equally sized gradients, and if using

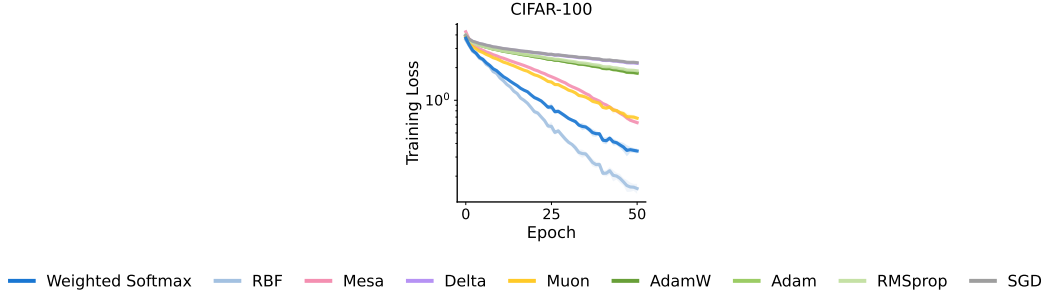


Figure 8: Training loss for Weighted Softmax & RBF decreases much faster than that of the other models.

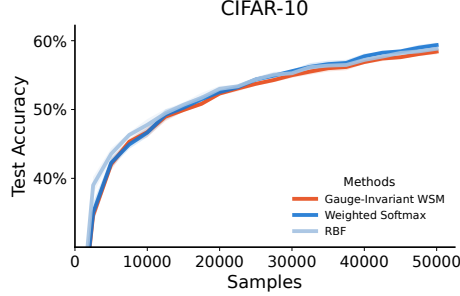


Figure 9: Test accuracy of non-parametric methods (Weighted Softmax, Gauge-Invariant WSM & RBF) as a function of the number of samples seen during training.

advanced optimizers such as Adam [38], first and second moments, making training significantly more expensive and slower than by learning the Weighted Softmax module by growing.

G Further Experimental Results

G.1 Training loss on CIFAR-100

Complementary to Figure 2/Middle where we report the train loss on CIFAR-10, we here report the respective numbers for CIFAR-100 in Figure 8.

G.2 Gauge-Invariant Weighted Softmax

We compare the Gauge-Invariant Weighted Softmax (Algorithm 2) with the standard Weighted Softmax and RBF in the single-epoch setting of Fig. 2/Right: non-augmented CIFAR-10, batch size 16, $L = 2$ residual blocks, $d_e = 3072$, and RMSNorm without learned scale. For Weighted Softmax and RBF we use the best configurations from the respective sweeps: Weighted Softmax with $T = 6$, $\eta = 0.003$, $\beta = 3$; RBF with temperature 20 and learning rate 10. For the Gauge-Invariant variant we use $T = 5$, $\alpha = 10$, $\eta = 0.03$, $\beta = 100$. Fig. 9 shows mean and standard deviation over seeds. All three methods reach similar final test accuracy (Weighted Softmax $59.4 \pm 0.4\%$, RBF $58.8 \pm 0.2\%$, Gauge-Invariant $58.4 \pm 0.4\%$). Guaranteeing non-negative weights by the choice of gauge, rather than by clipping, therefore performs comparably and removes the need for the clipping heuristic. Given this similar behavior, we decided for the results in the main text of the paper to only analyze standard weighted softmax RCDL layers.

H Further Related Work and Discussions

Here we provide further related work and discussions. At a high-level, retrieval-centric deep learning (RCDL) also shares conceptual similarities with existing “differentiable memory models” such as Differentiable Neural Computer [59, 60], Memory Networks [61, 62] and Neural Episodic Control [63] (see also more recent work in the context of language models, such as Berges et al. [64], Eyuboglu

et al. [65]); with a crucial difference that RCDL’s memory retrieval operates on the training datapoints. Its growing behavior, recruiting “resources” (new key/value allocations) as the system processes increasingly more training data (conceptually to infinity, with practical hardware limitations) is somewhat also reminiscent of Turing machines [66] as is the case for the transformer at runtime [6] (see also how Behrouz et al. [67] adopt a similar principle to recurrent nets).

RCDL also differs from “deep kernel learning” [68] which uses a deep NN as a learnable pre-processor to produce features for a single-layer Gaussian Process (GP); as well as from a special variant of modern Hopfield-lookup layers [69] that stores the static training data and retrieves their labels via softmax attention. This comparison also provides an interesting contrast to infinite-width deep learning theories, where NNs asymptotically reduce to a kernel machine, such as the neural tangent kernel (NTK; Jacot et al. [70], Lee et al. [71]); in that regime, a central question is whether deep NNs meaningfully learn representations or operate in a lazy training regime with fixed random features [72]. In contrast, RCDL learns representations in an NN that is explicitly structured as a (deep) kernel machine—an intriguing future work may be to study RCDL through the lens of NTK/GP.

The challenge of expanding model capacity has antecedents in early constructive algorithms [73, 74], which dynamically allocate new units to reduce error. In modern deep learning, this concept evolved into architectures like “progressive NNs” [75], which instantiate new columns for novel tasks to prevent forgetting. In fact, the idea of growing networks is rather common in continual learning (see, e.g., Yoon et al. [76]), as well as in architecture search/optimization (see, e.g., Stanley and Miikkulainen [77], Evci et al. [78], Butkus et al. [79]). RCDL offers a distinct path: unlike heuristics that add coarse-grained parametric modules, RCDL grows by accumulating fine-grained nonparametric key-value pairs, resulting in capacity that scales linearly with training data. Crucially, while this growth rate is fixed, the update rule itself is strictly principled: we do not add units arbitrarily, but rather derive the addition of each new key-value pair as a precise step of functional gradient descent on the global loss.

“Retrieval” has also received recent popularity in the context of large language models (LLMs) [80, 81] as a method to complement the deficiency of LLMs’ knowledge stored in their fixed-size weight matrices. In a sense, RCDL integrates such a process *within* the network itself; store the entire internet in the NN itself, and “compression” occurs at inference time through attention that recombines features—a slogan here is “*store the entire internet in a growing neural net, compression can wait!*”.