Computer Science and Artificial Intelligence Laboratory

Technical Report

MIT-CSAIL-TR-2010-051
CBCL-292

November 19, 2010

Generalization and Properties of the Neural Response

Jake Bouvrie, Tomaso Poggio, Lorenzo Rosasco, Steve Smale, and Andre Wibisono

massachusetts institute of technology, cambridge, ma 02139 usa — www.csail.mit.edu

Generalization and Properties of the Neural Response

Jake Bouvrie[*,†], Tomaso Poggio[†], Lorenzo Rosasco[†,◇], Steve Smale[‡], Andre Wibisono[†]

∗ – *Duke University*

† – *CBCL, McGovern Institute, MIT*

‡ – *City University of Hong Kong and University of California, Berkeley*

◇ – *Italian Institute of Technology*

jvb@mit.edu, tpoggio@mit.edu, lrosasco@mit.edu, smale@cityu.edu.hk, wibisono@mit.edu

November 16, 2010

Abstract

Hierarchical learning algorithms have enjoyed tremendous growth in recent years, with many new algorithms being proposed and applied to a wide range of applications. However, despite the apparent success of hierarchical algorithms in practice, the theory of hierarchical architectures remains at an early stage. In this paper we study the theoretical properties of hierarchical algorithms from a mathematical perspective. Our work is based on the framework of hierarchical architectures introduced by Smale et al. in the paper "Mathematics of the Neural Response", *Foundations of Computational Mathematics*, 2010. We propose a generalized definition of the neural response and derived kernel that allows us to integrate some of the existing hierarchical algorithms in practice into our framework. We then use this generalized definition to analyze the theoretical properties of hierarchical architectures. Our analysis focuses on three particular aspects of the hierarchy. First, we show that a wide class of architectures suffers from range compression; essentially, the derived kernel becomes increasingly saturated at each layer. Second, we show that the complexity of a linear architecture is constrained by the complexity of the first layer, and in some cases the architecture collapses into a single-layer linear computation. Finally, we characterize the discrimination and invariance properties of the derived kernel in the case when the input data are one-dimensional strings. We believe that these theoretical results will provide a useful foundation for guiding future developments within the theory of hierarchical algorithms.

Contents

1 Introduction **3**
- 1.1 Background and Motivation . . . 3
- 1.2 Contributions and Organization of this Paper . . . 4

2 A Theoretical Framework of Hierarchical Architectures **5**
- 2.1 Filtering and Pooling . . . 6
  - 2.1.1 Filtering Operations . . . 6
  - 2.1.2 Pooling Functions . . . 7
- 2.2 Hierarchy of Function Patches and Transformations . . . 8
- 2.3 Templates . . . 9
- 2.4 Neural Response and Derived Kernel . . . 10
- 2.5 Examples of Architectures . . . 12

3 Range Compression **13**
3.1 Setting and Preliminaries . . . 14
3.1.1 Nonnegative Architectures . . . 14
3.1.2 Strongly Bounded Pooling Functions . . . 15
3.1.3 Dynamic Range of the Derived Kernel . . . 16
3.2 Range Compression: Normalized Kernel Functions . . . 16
3.3 Range Compression: Normalized Pooling Functions . . . 17
3.4 Empirical Results and Possible Fixes to Range Compression . . . 21
3.5 Discussion . . . 24

4 Linear Architectures **24**
4.1 Rank Constraint . . . 25
4.2 Basis Independence . . . 25

5 Analysis in a One-Dimensional Case **26**
5.1 Setting and Preliminaries . . . 27
5.2 Reversal Invariance of the Derived Kernel . . . 28
5.2.1 Reversal Invariance from the Initial Kernel . . . 29
5.2.2 Reversal Invariance from the Transformations . . . 29
5.2.3 Reversal Invariance from the Templates . . . 30
5.2.4 Impossibility of Learning Reversal Invariance with Exhaustive Templates . . . 31
5.3 Equivalence Classes of the Derived Kernel . . . 31
5.4 Mirror Symmetry in Two-Dimensional Images . . . 32
5.5 Discussion . . . 33

6 Conclusions and Open Questions **34**

A Appendix: A Theoretical Framework of Hierarchical Architectures **36**

B Appendix: Range Compression **36**
B.1 Setting and Preliminaries . . . 36
B.1.1 Strongly Bounded Pooling Functions . . . 36
B.1.2 Dynamic Range of the Derived Kernel . . . 36
B.2 Range Compression: Normalized Kernel Functions . . . 40
B.3 Range Compression: Normalized Pooling Functions . . . 41

C Appendix: Linear Architectures **43**
C.1 Rank Constraint . . . 43
C.2 Basis Independence . . . 44

D Appendix: Analysis in a One-Dimensional Case **45**
D.1 Setting and Preliminaries . . . 45
D.2 Reversal Invariance of the Derived Kernel . . . 46
D.2.1 Reversal Invariance from the Initial Kernel . . . 46
D.2.2 Reversal Invariance from the Transformations . . . 47
D.2.3 Reversal Invariance from the Templates . . . 47
D.2.4 Impossibility of Learning Reversal Invariance with Exhaustive Templates . . . 48
D.3 Equivalence Classes of the Derived Kernel . . . 50
D.4 Mirror Symmetry in Two-Dimensional Images . . . 54

References **56**

1 Introduction

One of the remarkable powers of the human brain is its ability to learn and generalize from only a few examples, a skill generally attributed to the structure of the cortex. Drawing inspiration from the hierarchical structure of the visual cortex, many hierarchical algorithms have been proposed to improve the performance on image classification tasks. In recent years, the successes of these initial algorithms have led to applications of hierarchical algorithms in a wide range of areas. However, the reasons for these successes are poorly understood. With the rapid empirical development of hierarchical learning algorithms, a commensurate theory is needed to explain the phenomena encountered in practice, as well as to guide the direction of future research. Smale et al. [56] introduced a theoretical framework of hierarchical architectures that enables hierarchical algorithms in practice to be studied from a theoretical perspective. The main objective of this paper is to continue the work of Smale et al. [56] in laying the theoretical foundations of hierarchical architectures.

1.1 Background and Motivation

Classical kernel methods [53] such as regularization networks [14], splines [60], and support vector machines [12] can be described as single-layer networks because their inputs and outputs are generally related by a single computation of kernel functions. In contrast, hierarchical algorithms are characterized by a multi-layer architecture, where each layer of the architecture is performing similar computations consisting of the filtering and pooling operations. At each layer, the filtering operation matches the input data (or encodings from the previous layer) against a given set of templates, while the pooling operation combines the outputs of the filtering step into a single value. The alternating filtering and pooling operations are repeated through the hierarchy to produce an encoding of the input data, that we can pass to a classifier or use for further processing.

The recent development in hierarchical algorithms is to a large extent motivated by the advances in neuroscience that lead to our improved understanding of the organization and functions of the brain. Hubel and Wiesel [25] proposed that the early stages of the mammalian visual cortex are organized in a hierarchy of simple and complex cells. The simple cells are selective to certain sizes or orientations, while the complex cells are more invariant to small changes in the stimuli. As the input stimuli are processed through the layers of the visual cortex, the interleaving of the simple and complex cells results in a more complex representation of the stimuli. Starting with the work of Fukushima [16], many hierarchical algorithms that try to replicate the hierarchical structure of the visual cortex have been proposed [22, 31, 37, 50, 54]. The alternating filtering and pooling steps in the algorithms mimic the interconnection between the simple and complex cells in the visual cortex.

Hierarchical algorithms have been applied to a wide range of problem domains, including image classification [29, 48, 59], image segmentation [58], action recognition from video sequences [17, 27], speech recognition [9, 15, 38], dimensionality reduction [23, 52], robotics [20, 34], natural language processing [11, 42], language identification [63], and even seizure detection [40, 41], with encouraging results. Hierarchical algorithms have been shown to consistently outperform classical shallow networks in practice [21, 29, 33]. However, despite the apparent empirical success of hierarchical algorithms, so far there is little understanding as to why these algorithms work as well as they do.

Several arguments have been proposed to explain the theoretical properties of hierarchical algorithms, in particular regarding their representational powers. A classical result in the neural networks literature states that neural networks with one hidden layer and sufficiently many hidden units can approximate arbitrary measurable functions [24]. Similarly, neural networks with sufficiently many layers and only one hidden unit in every layer have been shown to be universal classifiers [51]. More recently, Sutskever and Hinton [57] showed that deep belief networks with limited width are universal approximators for distributions over binary vectors, and Le Roux and Bengio [30] further improved the upper bound on the number of parameters required to achieve the universal approximation. Finally, Bengio [3] and Bengio and LeCun [5] argued that a deep architecture is necessary to find an efficient representation of

a highly-varying function.

Nevertheless, many fundamental questions remain unanswered and the work on hierarchical algorithms remains largely empirical [4, 13, 29]. For example, one of the basic questions regarding hierarchical algorithms is how to choose the parameters to build a good architecture. This turns out to be a difficult question, in part because the answer might depend on the specific task at hand, and the theoretical answer to this question has eluded our understanding thus far. Consequently, several authors have explored this question empirically [26, 45].

Another example of a major area of empirical work in hierarchical algorithms is related to the invariance properties of the architecture. The repeated operations of pooling over local regions intuitively lead to some invariance properties in the hierarchy. Indeed, the trade-off between the discrimination and invariance properties in the architecture is central to the development of hierarchical algorithms in practice. But can we formalize this intuition? Again, this question is surprisingly difficult to answer theoretically, and invariance in hierarchical algorithms has generally only been analyzed empirically [19, 28].

Admittedly, empirical approaches can provide good results and intuitions about how hierarchical algorithms work in practice. However, empirical works alone are not sufficient because they cannot provide satisfying explanations about phenomena that occur in nature (for example, why the visual cortex is structured as a hierarchy), or even in practice (why one algorithm works better than another). We believe that in order to advance our understanding of hierarchical algorithms, a theory is needed. A theoretical analysis of hierarchical algorithms can provide an insight into the behavior and characteristics of hierarchical architectures. Furthermore, theoretical analysis can help identify problems that occur in practice and propose solutions.

One of the main difficulties in performing a theoretical study on hierarchical algorithms is the lack of a common framework. Existing hierarchical algorithms in practice are typically quite complex with specific details that differ from algorithms to algorithms, thus preventing us from translating results on one algorithm into another. In an attempt to overcome this difficulty, Smale et al. [56] proposed a theoretical framework of hierarchical architectures. The elegant formalization of this framework strikes a delicate balance between the complexity of the model and the faithfulness to neuroscience and hierarchical algorithms in practice.

The central objects of interest in the framework that Smale et al. proposed are the *neural response* and *derived kernel*. At each layer, the neural response is an encoding of the input data, and the derived kernel is a similarity measure on the data that is defined in terms of the neural responses. The neural response and derived kernel at each layer are constructed recursively using an alternating process of filtering and pooling, much like the principal operations in hierarchical algorithms that we previously described.

The framework of neural response and derived kernel that Smale et al. introduced is based primarily on the feedforward hierarchical model [50, 54]. However, this framework eliminates several implementation and engineering details from the algorithms and only preserves the essential components that characterize hierarchical architectures. As such, this framework is amenable to theoretical analysis. Furthermore, this framework also has the potential to be generalized to include other hierarchical algorithms in practice.

Indeed, preliminary results on the analysis of hierarchical architectures using the framework of the neural response and derived kernel are promising [56]. Further analysis on the invariance properties of the derived kernel from a group-theoretic perspective is carried out in [8, 10]. However, the introduction of this framework also presents a challenge of whether the framework of the neural response and derived kernel can continue to live up to its promise to bridge the gap between the theory and practice of hierarchical algorithms. This is the question that we seek to address in this paper.

1.2 Contributions and Organization of this Paper

In this paper we study the theoretical properties of hierarchical architectures using the framework introduced by Smale et al. [56]. We present our results in the main sections, but we defer all proofs and more technical discussion to the appendix corresponding to each section. We summarize our contributions as follows.

Section 2 – A theoretical framework of hierarchical architectures. We describe the theoretical framework of hierarchical architectures and propose a generalized definition of the neural response and derived kernel. This generalized definition allows us to study a wider class of hierarchical algorithms in our framework, including the feedforward hierarchical architecture [50, 54] and convolutional neural network [26, 31, 32].

Section 3 – Range compression. We show that a wide class of hierarchical architectures suffers from range compression; essentially, the derived kernel becomes increasingly saturated at each layer. Range compression presents a problem in practice because the saturation effect is severe enough that it affects the performance of the architecture, even when the architecture has only a few layers. We propose several methods to mitigate the saturation effect and check that these correction methods restore the performance of the architecture in practice. This result lends insight into how the interaction between certain parameter choices in the architecture can influence its empirical performance.

Section 4 – Linear architectures. We study the limitations and properties of linear architectures. We show that the complexity of a linear architecture is constrained by the complexity of the first layer, and in some cases the architecture collapses into a single-layer linear computation. Furthermore, we also show that when we use an orthonormal basis as templates for the filtering operation, the derived kernel is independent of the basis.

Section 5 – Analysis in a one-dimensional case. We extend the discrimination and invariance analysis of Smale et al. [56] in the case when the input data are one-dimensional strings. We demonstrate several ways to build a reversal invariant derived kernel by choosing the appropriate parameters in the model. We also show that when we use an exhaustive set of templates, it is impossible to learn reversal invariance in the architecture. Finally, we characterize the equivalence classes of the derived kernel in the case when we are working with a discriminative architecture. We can interpret these equivalence classes as the intrinsic translation invariance in the architecture.

2 A Theoretical Framework of Hierarchical Architectures

In this section we propose a generalization of the theoretical framework of hierarchical architectures that Smale et al. introduced in [56]. We extend the definition of the neural response and derived kernel to allow the use of general pooling functions, kernel functions, and templates with suitable admissibility properties. This generalized framework enables us to study a wider class of hierarchical architectures, including several hierarchical algorithms in practice such as the feedforward architecture [50, 54] and convolutional neural networks [26, 31, 32]. A theoretical analysis on this generalized framework can thus be more applicable to problems in practice, and ultimately guide the development of more efficient hierarchical learning machines.

The basic assumption underlying hierarchical learning algorithms is that each input can be decomposed into a hierarchy of parts of increasing size with increasing semantic complexity. Given a representation of the smallest parts, the hierarchical architecture recursively builds at each layer a representation of the next larger parts by a combination of filtering and pooling operations. The filtering operation measures the similarity between the input representations and a given set of templates, while the pooling operation combines the filtered outputs into a single value. Intuitively, the filtering step induces discrimination while the pooling step induces invariance in the architecture. Thus, the alternating applications of the filtering and pooling operations yield a complex representation of the input data with nontrivial discrimination and invariance properties. This trade-off between discrimination and invariance is widely seen as a key property of hierarchical architectures that makes hierarchical algorithms work well in practice.

2.1 Filtering and Pooling

We first formalize the two basic operations performed at each layer in the architecture: filtering and pooling.

2.1.1 Filtering Operations

We propose to describe the filtering operation using a reproducing kernel. Recall that given a space X, a *reproducing kernel* is a symmetric and positive definite function $K\colon X \times X \to \mathbb{R}$. A map Φ from X to a Hilbert space $\mathcal{H}$ is called a *feature map associated to* K if

$$\big\langle \Phi(x),\, \Phi(y) \big\rangle_{\mathcal{H}} = K(x,y) \qquad \text{for all } x, y \in X \ .$$

It can be shown that any such map Φ defines a reproducing kernel, and conversely, given a reproducing kernel it is always possible to construct an associated feature map [1].

In the class of hierarchical architectures that we are considering, the inputs to the filtering operation will in general have different lengths from layer to layer. To address this issue, we define the kernel K on a sufficiently large space that contains all the possible inputs, and work with the restrictions of K on the appropriate domains of the inputs. Specifically, we begin by considering a kernel $K\colon \ell^2 \times \ell^2 \to \mathbb{R}$, where ℓ^2 is the space of square summable sequences. For every $n \in \mathbb{N}$, we can consider the induced kernel $K_n\colon \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ given by

$$K_n(x,y) = K\big(\epsilon_n(x),\, \epsilon_n(y)\big) \qquad \text{for all } x, y \in \mathbb{R}^n \ ,$$

where $\epsilon_n\colon \mathbb{R}^n \to \ell^2$ is the embedding of $\mathbb{R}^n$ as an n-dimensional subspace of ℓ^2,

$$\epsilon_n(x) = (x_1, \dots, x_n, 0, 0, \dots) \in \ell^2 \qquad \text{for } x = (x_1, \dots, x_n) \in \mathbb{R}^n \ .$$

Equivalently, K_n is the restriction of K on the subspace $\epsilon_n(\mathbb{R}^n) \times \epsilon_n(\mathbb{R}^n) \subseteq \ell^2 \times \ell^2$, and is thus itself a reproducing kernel. The induced kernels K_n are hereafter denoted by K since the dependence on n will be clear from context. Moreover, all inner products and norms will denote the standard inner product and norm in ℓ^2 unless specified otherwise. Some examples of kernel functions of interest are given in Table 1.[a]

Name	Expression
Inner product	$K(x,y) = \langle x, y \rangle$
Normalized inner product	$K(x,y) = \dfrac{\langle x, y \rangle}{\|x\|\,\|y\|}$
Gaussian	$K(x,y) = e^{-\gamma \|x-y\|^2}$

Table 1: Examples of kernel functions.

We will also often be working with a normalized kernel. We say that a kernel $K\colon \ell^2 \times \ell^2 \to \mathbb{R}$ is *normalized* if $K(x,x) = 1$ for all $x \in \ell^2$. For example, the normalized inner product and Gaussian kernels given in Table 1 are normalized. Clearly the normalization of K is equivalent to the condition that the feature map associated to K takes values in the unit sphere in ℓ^2, that is, $\|\Phi(x)\| = 1$ for all $x \in \ell^2$. Furthermore, by the Cauchy–Schwarz inequality it is easy to see that if K is normalized, then

[a] For the normalized inner product kernel, we take the domain of K to be $(\ell^2 \setminus \{0\}) \times (\ell^2 \setminus \{0\})$.

$-1 \leq K(x,y) \leq 1$ for all $x, y \in \ell^2$. Finally, we note that given a reproducing kernel $K\colon \ell^2 \times \ell^2 \to \mathbb{R}$, we can construct a normalized reproducing kernel $\widehat{K}\colon \ell^2 \times \ell^2 \to \mathbb{R}$ by

$$\widehat{K}(x,y) = \frac{K(x,y)}{\sqrt{K(x,x)\,K(y,y)}} \quad \text{for all } x, y \in \ell^2 \ ,$$

which corresponds to normalizing the feature map to be of unit length,

$$\widehat{\Phi}(x) = \frac{\Phi(x)}{\|\Phi(x)\|} \quad \text{for all } x \in \ell^2 \ .$$

2.1.2 Pooling Functions

The pooling operation can be described as a function that summarizes the content of a sequence of values with a single value, similar to aggregation functions used in voting schemes and database systems [2]. As in the case of the filtering, the inputs to the pooling operation will in general have different lengths in different layers. Following the treatment in [2], we define a *pooling function* to be a function

$$\Psi\colon \mathbb{R}^* \to \mathbb{R} \ ,$$

where $\mathbb{R}^* = \bigcup_{n\in\mathbb{N}} \mathbb{R}^n$ is the set of all finite-length sequences. We note that in general, the actions of Ψ on input values with different lengths do not have to be related to each other. However, in the examples that we are going to consider, Ψ is defined by an underlying formula that can be applied to input sequences of arbitrary lengths. For example, the average pooling function can be described by the following formula,

$$\Psi(\alpha) = \frac{1}{n}\sum_{i=1}^{n} \alpha_i \quad \text{if } \alpha \in \mathbb{R}^n \ .$$

In the definition of the neural response, we will often work with the case when the input to the pooling function is a sequence indexed by a finite set H. In an abuse of notation, given one such sequence $\alpha\colon H \to \mathbb{R}$, we will write

$$\Psi\big(\alpha(h)\big)$$

to denote the output of the pooling operation on the input $\big(\alpha(h)\big)_{h\in H} \in \mathbb{R}^{|H|}$. Moreover, given a pooling function $\Psi\colon \mathbb{R}^* \to \mathbb{R}$ and a function $\zeta\colon \mathbb{R} \to \mathbb{R}$, we write $\Psi \circ \zeta$ to denote the pooling function obtained by first applying ζ componentwise on the input and then applying Ψ, that is,

$$(\Psi \circ \zeta)(\alpha) = \Psi\big((\zeta(\alpha_i))_{i=1}^n\big) \quad \text{for all } \alpha = (\alpha_1, \ldots, \alpha_n) \in \mathbb{R}^n,\ n \in \mathbb{N} \ .$$

This form of pooling function is particularly important when ζ is a sigmoid function (see Example 2.12 and Section 3.3). Several examples of common pooling functions are given in Table 2.

In anticipation of the upcoming development, we will at times impose the following condition on the pooling function to ensure that the construction of the neural response and derived kernel is well defined.

Definition 2.1 (Weakly bounded pooling function)**.** *A pooling function $\Psi\colon \mathbb{R}^* \to \mathbb{R}$ is weakly bounded if for every $n \in \mathbb{N}$ there exists a constant $C_n > 0$ such that*

$$|\Psi(\alpha)| \leq C_n \, \|\alpha\| \quad \textit{for all } \alpha \in \mathbb{R}^n \ .$$

We note that weak boundedness is precisely the boundedness condition when Ψ is a linear functional. In particular, weak boundedness is a very mild condition and is satisfied by most reasonable pooling functions. For instance, all the pooling functions in Table 2 are weakly bounded. Clearly the max and ℓ^∞-norm pooling functions are weakly bounded with $C_n = 1$. Moreover, by using the Cauchy–Schwarz inequality we can show that the average and ℓ^1-norm pooling functions are also weakly bounded with constants $C_n = 1/\sqrt{n}$ and $C_n = \sqrt{n}$, respectively.

Name	Expression
Average	$\Psi(\alpha(h)) = \frac{1}{\lvert H\rvert}\sum_{h\in H}\alpha(h)$
ℓ^1-norm	$\Psi(\alpha(h)) = \sum_{h\in H}\lvert\alpha(h)\rvert$
Max	$\Psi(\alpha(h)) = \max_{h\in H}\alpha(h)$
ℓ^∞-norm	$\Psi(\alpha(h)) = \max_{h\in H}\lvert\alpha(h)\rvert$

Table 2: Examples of pooling functions.

On the other hand, weak boundedness also imposes some restrictions on the behavior of Ψ. For example, by taking $\alpha = 0$ in Definition 2.1 we see that a weakly bounded Ψ must satisfy $\Psi(0) = 0$. Furthermore, the assumption that Ψ is bounded above by a norm implies that Ψ must have a homogeneous growth rate. For instance, the function $\Psi(\alpha) = \|\alpha\|$ is weakly bounded while $\Psi(\alpha) = \|\alpha\|^2$ is not, since we do not have an upper bound on $\|\alpha\|$.

2.2 Hierarchy of Function Patches and Transformations

We now describe the fundamental structures of the inputs of the hierarchical architecture. This mathematical framework formalizes the decomposability assumption underlying the hierarchical structure of the architecture, that each input is composed of parts of increasing size.

Functions. One convention that we maintain in our framework is that we represent every input as a *function* f from a domain v to a set of values Ω. For example, in the case of vision, we can represent an image as a real-valued function over a receptive field, which can be taken to be a square region in $\mathbb{R}^2$. When we are working with (grayscale) images in computers, an image of size $k \times k$ pixels is a function $f\colon \{1,\dots,k\}\times\{1,\dots,k\} \to [0,1]$. Finally, in the case of one-dimensional strings (see Section 5), a string of length k is a function $f\colon \{1,\dots,k\} \to S$, where S is a finite alphabet.

Hierarchical architecture. A hierarchical architecture with $n \in \mathbb{N}$ layers consists of a nested sequence of *patches* $v_1 \subseteq v_2 \subseteq \cdots \subseteq v_n$. At each layer $1 \le m \le n$ we have a *function space* $\mathrm{Im}(v_m)$ consisting of the functions $f\colon v_m \to \Omega$ that we will be working with. The nested structure of the patches corresponds to the assumption that the functions at lower layers are less complex than the functions on higher layers. Note that we do not require $\mathrm{Im}(v_m)$ to contain all possible functions $f\colon v_m \to \Omega$, nor do we assume any algebraic or topological structures on $\mathrm{Im}(v_m)$.

Transformations. The connection between adjacent patches v_m and v_{m+1} is provided by a set H_m of *transformations* $h\colon v_m \to v_{m+1}$. Each such transformation h specifies how to embed a patch in a larger patch. Equivalently, the transformations specify how to decompose a patch into a collection of smaller patches, as well as how to decompose a function into function patches. Given a function $f\colon v_{m+1} \to \Omega$ in $\mathrm{Im}(v_{m+1})$ and a transformation $h\colon v_m \to v_{m+1}$, we can restrict f with h to obtain a function patch $f\circ h\colon v_m \to \Omega$. In order for this restriction operation to be well defined, we need the following assumption.

Assumption 1. *For $1 \le m \le n-1$, we have $f \circ h \in Im(v_m)$ if $f \in Im(v_{m+1})$ and $h \in H_m$.*

Note that Assumption 1 is very mild and can be easily satisfied in practice. For example, we can take $\mathrm{Im}(v_m)$ to be the set of all possible functions $f\colon v_m \to \Omega$. Another way to construct function spaces that satisfy Assumption 1 is to start with the function space $\mathrm{Im}(v_n)$ at the highest layer and recursively define the function spaces at the lower layers by restricting the functions,

$$\mathrm{Im}(v_m) = \{f \circ h \mid f \in \mathrm{Im}(v_{m+1}),\ h \in H_m\} \quad \text{for } 1 \le m \le n-1 \ .$$

Examples of transformations in the case of images include translations, rotations, or reflections. Note further that as in the case of function spaces, we do not require H_m to contain all possible transformations $h\colon v_m \to v_{m+1}$, nor do we assume any structures on H_m. However, in this paper we always make the following assumption.

Assumption 2. *For $1 \le m \le n-1$, the set of transformations H_m is nonempty and finite.*

2.3 Templates

A key semantic component in the hierarchical architecture is a dictionary of *templates* that is usually learned from data. The templates will play the role of the filters that we are comparing against in the filtering operation. The templates also provide a connection between the architecture and the underlying distribution of the input data. For example, one way to generate templates is to sample from the probability distribution on the function space. As an illustration, in the case of vision, the templates can be thought of as representing the primary images that we want the architecture to learn, either because they are the predominant images in our world, or because they can serve as a building block to represent other images.

To describe the concept of templates, we first need the following definition of admissible sets.

Definition 2.2 (Admissible set)**.** *Given a Hilbert space $\mathcal{H}$, a set $\mathcal{T} \subseteq \mathcal{H}$ is admissible if there is a constant $C > 0$ such that*

$$\sum_{\tau \in \mathcal{T}} \langle \alpha, \tau \rangle_{\mathcal{H}}^2 \le C \|\alpha\|_{\mathcal{H}}^2 \quad \text{for all } \alpha \in \mathcal{H} \ .$$

Clearly any finite set is admissible. When $\mathcal{H} = \ell^2$, an example of an infinite admissible set is when we take $\mathcal{T}$ to be an orthonormal basis $(\tau_i)_{i=1}^{\infty}$ of ℓ^2, and in fact in this case we have $\sum_{i=1}^{\infty} \langle \alpha, \tau_i \rangle^2 = \|\alpha\|^2$. The admissibility condition is also satisfied by more general dictionaries, for example if $\mathcal{T}$ is a frame for $\mathcal{H}$.

Given the definition above, we can formalize the notion of templates. Recall that given a normed vector space V and a non-empty subset $W \subseteq V$, the *closed linear span* of W, denoted by $\overline{\mathrm{Span}}(W)$, is the closure of the linear span of W in V.

Definition 2.3 (Templates)**.** *Consider the function space $Im(v_m)$ at some layer m, and let $F\colon Im(v_m) \to \mathcal{H}$ be a feature map, where $\mathcal{H}$ is a Hilbert space. We say that $\mathcal{T}_m$ is an m-th layer set of templates associated to F if $\mathcal{T}_m$ is a nonempty admissible set in the closed linear span of $\{F(t) \mid t \in Im(v_m)\}$.*

We note that the admissibility condition is needed for theoretical reasons to ensure that we can work with an arbitrarily large, possibly infinite, set of templates. In practice, the number of templates is always finite and the admissibility condition is always satisfied.

As is clear from the definition, there are two inherently different forms of templates: those that are the encoded functions in $\mathrm{Im}(v_m)$, and those that are linear combinations of the encoded functions. Given a set of templates $\mathcal{T}_m$ associated to F, we distinguish these two cases as follows.

1. **Templates of the first kind.** We say that $\mathcal{T}_m$ is a set of *templates of the first kind* if every template $\tau \in \mathcal{T}_m$ is of the form

   $$\tau = F(t)$$

   for some $t \in \mathrm{Im}(v_m)$. This is the dictionary used in [56] when the feature map F is the neural response (see Definition 2.6) and will be our primary example in this paper.

2. **Templates of the second kind.** Otherwise, we say that $\mathcal{T}_m$ is a set of *templates of the second kind* if it is not of the first kind. In this case every template $\tau \in \mathcal{T}_m$ is a (possibly infinite) linear combination of the encoded functions $F(t)$, for some $t \in \mathrm{Im}(v_m)$. For example, we can take $\mathcal{T}_m$ to be a set of convex combinations of templates of the first kind (see Section 3.1), or we can take $\mathcal{T}_m$ to be an orthonormal basis of $\overline{\mathrm{Span}}\{F(t) \mid t \in \mathrm{Im}(v_m)\}$ (see Section 4.2).

Remark 2.4 (Templates learning)**.** In addition to the examples of templates given in the discussion above, we can also use other learning schemes to obtain the dictionary of templates. For example, given a set $\mathcal{T}_m$ of templates of the first kind, we can extract a set $\mathcal{T}'_m$ of templates of the second kind by using techniques such as PCA, ICA, sparse coding [36, 39], and nonnegative matrix factorization [35]. Therefore, our analytical framework provides us with an abstraction to treat the different learning methods in practice as a change of templates for the filtering operation at each layer.

Remark 2.5 (Multiset of Templates)**.** In the remainder of this paper we treat the template set as a multiset; that is, we allow the possibility of having two identical templates. In particular, when we are using templates of the first kind, we start with a set of function patches $T_m \subseteq \mathrm{Im}(v_m)$ and take the templates to be $\mathcal{T}_m = \{\widehat{N}_m(t) \mid t \in T_m\}$. In this case, even when we have two distinct function patches $t_1 \neq t_2$ with $\widehat{N}_m(t_1) = \widehat{N}_m(t_2)$, we still keep both templates in $\mathcal{T}_m$, so that $\mathcal{T}_m$ has the same number of elements as T_m.

2.4 Neural Response and Derived Kernel

Given the settings described in the previous sections, we are now ready to define the central objects of interest in a hierarchical architecture, namely the neural response and derived kernel. At each layer m, the neural response is an encoding of the functions in $\mathrm{Im}(v_m)$, and the derived kernel is a similarity measure between between functions in $\mathrm{Im}(v_m)$, defined in terms of the neural response encodings. Naturally, since we are working with a hierarchical architecture, the neural response and derived kernel are defined recursively. In the definition below, $\ell^2(\mathcal{T}_{m-1})$ denotes the space of all square summable sequences $\alpha \colon \mathcal{T}_{m-1} \to \mathbb{R}$. Recall from our discussion in Section 2.1 that $\ell^2(\mathcal{T}_{m-1})$ can be embedded into ℓ^2.

Definition 2.6 (Neural response and derived kernel)**.** *Let $K \colon \ell^2 \times \ell^2 \to \mathbb{R}$ be a reproducing kernel with an associated feature map $\Phi \colon \ell^2 \to \ell^2$, let $\Psi \colon \mathbb{R}^* \to \mathbb{R}$ be a pooling function, and let $N_1 \colon Im(v_1) \to \ell^2$ be a feature map. The m-th layer neural response $N_m \colon Im(v_m) \to \ell^2(\mathcal{T}_{m-1})$ is defined as*

$$N_m(f)(\tau) = \Psi\big(\langle \Phi(N_{m-1}(f \circ h)),\, \tau \rangle\big) \quad \text{for all } f \in Im(v_m) \text{ and } \tau \in \mathcal{T}_{m-1} \ , \tag{1}$$

where the pooling operation is performed over $h \in H_{m-1}$, and $\mathcal{T}_{m-1}$ is a set of templates associated to $\Phi \circ N_{m-1}$. The m-th layer derived kernel $K_m \colon Im(v_m) \times Im(v_m) \to \mathbb{R}$ is given by

$$K_m(f,g) = K\big(N_m(f), N_m(g)\big) = \big\langle \Phi\big(N_m(f)\big),\, \Phi\big(N_m(g)\big) \big\rangle \quad \text{for all } f, g \in Im(v_m) \ .$$

We note that the template sets $\mathcal{T}_m$ at each layer are computed after we have constructed the neural response N_m. As we mentioned in Section 2.3, the primary example of templates that we are going to use is templates of the first kind. In this case we start with a set of function patches $T_m \subseteq \mathrm{Im}(v_m)$, and take the set of templates $\mathcal{T}_m$ to be

$$\mathcal{T}_m = \{\widehat{N}_m(t) \mid t \in T_m\} \ .$$

Then the definition of the neural response in (1) takes a more familiar form

$$N_m(f)(\tau) = \Psi\big(K_{m-1}(f \circ h,\, t)\big) \quad \text{for all } f \in \mathrm{Im}(v_m) \text{ and } \tau = \widehat{N}_{m-1}(t) \in \mathcal{T}_{m-1} \ ,$$

which is precisely the definition given in [56] in the case when Ψ is the max pooling function. The function patches in T_m can be either a fixed set of patterns that we want to use, for instance oriented bars or Gabor filters [44, 55] in the case of vision, or sampled from the probability distribution on $\mathrm{Im}(v_m)$.

When the template sets are finite, the neural response in Definition 2.6 is finite dimensional and well defined. When the template sets are infinite, however, the assumption on the weak boundedness of the pooling function together with the admissibility condition of the templates are crucial to ensure that the neural response is still well defined.

Proposition 2.7. *Suppose the template set $\mathcal{T}_m$ is infinite for some $1 \leq m \leq n-1$. If the pooling function Ψ is weakly bounded, then $N_{m+1}(f) \in \ell^2$ for all $f \in Im(v_{m+1})$.*

In light of the result above, throughout this paper we assume that the pooling function Ψ is weakly bounded whenever we are working with infinitely many templates.

Assumption 3. *If the template set $\mathcal{T}_m$ is infinite for some $1 \leq m \leq n-1$, then we assume that the pooling function $\Psi \colon \mathbb{R}^* \to \mathbb{R}$ is weakly bounded.*

Note that we do not make the assumption that Ψ is weakly bounded when the template sets are finite. This is not only because this assumption is unnecessary when we are working with finitely many templates, but also because this assumption will preclude some common settings that occur in practice. For example, the composite pooling function $\Psi \circ \zeta$ is not weakly bounded if $\zeta(0) \neq 0$ and Ψ is the ℓ^1-norm pooling function. However, this is precisely the situation in convolutional neural network when $\zeta = \sigma$ is the logistic function (see Example 2.12).

Remark 2.8 (Normalization)**.** Normalization is a procedure that standardizes the outputs of different layers in the architecture. Normalization is usually performed to enable us to meaningfully compare the results between different layers in the architecture, or between different architectures with potentially different number of layers. Hierarchical algorithms in practice typically have some normalization procedures, in one form or another [26, 45, 54].

Normalization is primarily performed either via the kernel or via the pooling function. Normalization via the kernel is achieved by using a normalized reproducing kernel, for example the normalized inner product or Gaussian kernel. On the other hand, normalization via the pooling function is achieved by using a pooling function that has a thresholding behavior, so that the values of the neural response and derived kernel can be controlled to lie in a certain range. We will consider these two methods of normalization in more detail in Section 3 and study their effects on the dynamic range of the derived kernel.

We note that in addition to the normalization methods that we describe here, there are also other ways to perform normalization that are employed in practice. For example, one normalization method that is suggested in practice is to use the normalized cross-correlation of the neural responses as the derived kernel [45].

Remark 2.9 (Extension)**.** In some architectures in practice, for example in the feedforward hierarchical model [54] and convolutional neural network [26], there is an intermediate step between the pooling and the filtering operations that we call the *extension step*. In this step, a function on an input patch is decomposed into smaller subpatches, each of which is encoded into a neural response independently following the procedure described in Definition 2.6. The results are then concatenated together to form the neural response of the original function. The kernel function on the concatenated neural response is taken to be the sum of the kernel functions on the individual components, and this recombined kernel function is used in Definition 2.6 to construct the neural response at the next layer.

More formally, we can describe the extension step in our framework as follows. At each layer m we take an intermediate patch w_m such that $v_m \subseteq w_m \subseteq v_{m+1}$. Consequently, we have two sets of transformations: H_{v_m,w_m} contains the transformations $h \colon v_m \to w_m$, and $H_{w_m,v_{m+1}}$ contains the transformations $h \colon w_m \to v_{m+1}$. Given a neural response $N_{v_m} \colon \mathrm{Im}(v_m) \to \ell^2$ on the patch v_m, we extend it to construct the neural response on the patch w_m,

$$N_{w_m} \colon \mathrm{Im}(w_m) \to \bigoplus_{h \in H_{v_m,w_m}} \ell^2 \cong \ell^2 \ ,$$

obtained by concatenating the neural responses of the smaller subpatches,

$$N_{w_m}(f) = \big(N_{v_m}(f \circ h_1),\ N_{v_m}(f \circ h_2),\ \ldots,\ N_{v_m}(f \circ h_k)\big) \quad \text{for all } f \in \mathrm{Im}(w_m)\ ,$$

where $h_i \in H_{v_m,w_m}$ for $1 \leq i \leq k = |H_{v_m,w_m}|$. The feature map Φ extends to this framework by acting on the individual components separately,

$$\Phi\big(N_{w_m}(f)\big) = \big(\Phi(N_{v_m}(f \circ h_1)),\ \Phi(N_{v_m}(f \circ h_2)),\ \ldots,\ \Phi(N_{v_m}(f \circ h_k))\big)\ .$$

Finally, the kernel function on w_m manifests as a linear combination of the kernel functions on the individual components,

$$K_{w_m}(f,g) = \big\langle \Phi(N_{w_m}(f)),\ \Phi(N_{w_m}(g)) \big\rangle = \sum_{h \in H_{v_m,w_m}} K_{v_m}(f \circ h,\ g \circ h) \quad \text{for all } f,g \in \mathrm{Im}(w_m)\ . \tag{2}$$

Having constructed K_{w_m}, we then proceed to define the neural response at the next layer, $N_{v_{m+1}}$, in terms of K_{w_m} by following Definition 2.6.

2.5 Examples of Architectures

The generalized definition of hierarchical architectures that we just described allows us to capture several hierarchical algorithms in practice in our framework. However, we note that in each case we are only considering a simplification of the algorithm that retains only the essential elements of the hierarchical structure.

Example 2.10. The original definition of the neural response and derived kernel proposed in [56] corresponds to the choice of the max pooling function, normalized inner product kernel, and templates of the first kind.

Example 2.11. The feedforward hierarchical model [50, 54] corresponds to the choice of the max pooling function and the Gaussian kernel. The templates are typically of the first kind. Given a template of the first kind $\tau \in \mathcal{T}_{m-1}$ corresponding to a function $t \in \mathrm{Im}(v_{m-1})$, the induced neural response can be written as

$$N_m(f)(\tau) = \max_{h \in H_{m-1}} e^{-\gamma \|N_{m-1}(f \circ h) - N_{m-1}(t)\|^2}\ .$$

An alternative architecture can be obtained by replacing the Gaussian kernel with the normalized inner product kernel, so that

$$N_m(f)(\tau) = \max_{h \in H_{m-1}} \frac{\big\langle N_{m-1}(f \circ h),\ N_{m-1}(t) \big\rangle}{\|N_{m-1}(f \circ h)\|\ \|N_{m-1}(t)\|}\ .$$

Example 2.12. The convolutional neural networks [26, 31, 32] use the pooling function

$$\Psi = \ell^1 \circ \sigma\ ,$$

where ℓ^1 denotes the (normalized) ℓ^1-norm pooling function and σ is a sigmoid function that is applied to each component of the input separately (see Definition 3.9). The kernel function is taken to be the inner product kernel, and the templates are typically of the second kind. The neural response corresponding to convolutional neural networks is then

$$N_m(f)(\tau) = \frac{1}{|H_{m-1}|} \sum_{h \in H_{m-1}} \big|\sigma\big(\langle N_{m-1}(f \circ h),\ \tau \rangle\big)\big|\ .$$

Example 2.13. The feedforward neural networks can be written as a hierarchical architecture where all the patches are of the same size, $v_1 = v_2 = \cdots = v_n$. In this case there is no decomposition of inputs and the set of transformations at each layer contains only the identity. The pooling function is a sigmoid nonlinearity, and does not result in any nontrivial invariance properties. The kernel function is taken to be the inner product kernel, so the corresponding neural response is

$$N_m(f)(\tau) = \sigma\big(\langle N_{m-1}(f),\ \tau \rangle\big)\ .$$

3 Range Compression

One property that applies to a wide class of hierarchical architectures is *range compression*. This property states that the dynamic range of the derived kernel is reduced at every layer, to the point that the derived kernel converges to a constant function when the number of layers becomes arbitrarily large. In effect, this loss of dynamic range means that all the input functions are eventually mapped to the same neural response, and hence the hierarchical architecture loses its discrimination property. Range compression has been observed in practice and is suspected to be responsible for poor classification performance in some tasks [18].

Broadly speaking, range compression is endemic in the class of architectures in which we are performing normalization on positive values, either via the kernel or the pooling function. If all the input values are positive, normalization introduces a tendency for the resulting values to shift away from 0. When quantified with the right measure, this drift amounts to exactly the loss of dynamic range that we just described. For example, we shall see that range compression is present in the following architectures:

1. The feedforward hierarchical model (Example 2.11) with the normalized inner product kernel and max pooling function.

2. The convolutional neural network (Example 2.12) with the inner product kernel, normalized ℓ^1-norm pooling with a componentwise sigmoid function, and a large class of templates of the second kind.

In this section we study the precise conditions on the architecture that lead to range compression. Our approach is to establish a layerwise lower bound for the compression rate of the derived kernel. This layerwise bound will yield the convergence of the derived kernel and neural response in the limit when the number of layers approaches ∞, as claimed above. We remark that our theoretical result is a conservative worst-case lower bound, and we observe in practice that the convergence rate is much faster.

In fact, the high rate at which range compression occurs poses a real problem in practice. Because the rate of convergence is fast enough, the derived kernel becomes *saturated* after only a small number of layers. That is, the effective range of the derived kernel becomes too small compared to the machine precision so that the kernel values become indistinguishable, thus rendering multilayer architectures unusable. This problem is particularly severe when we are working in a limited-precision environment, for example when we are using single precision floating points that GPUs commonly use, because then the saturation effect occurs earlier. Thus, this problem also prevents us from taking the full advantage of recent fast implementations of deep architectures using multipurpose GPU machines [43, 45, 47].

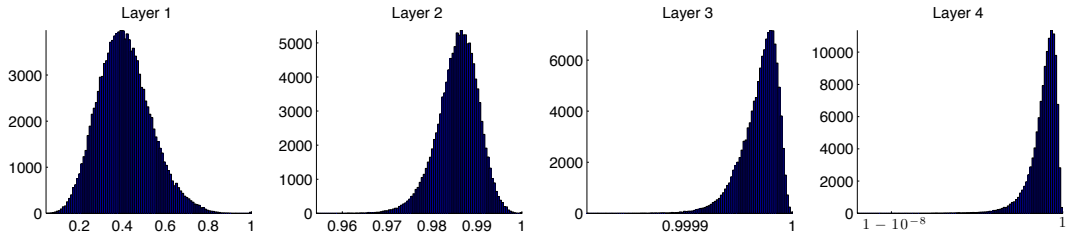


Figure 1: The effect of range compression on the sample distribution of $K_m(f,g)$ in an m-layer architecture, for $1 \le m \le 4$. Note the different scales on the plots.

To illustrate the range compression problem, Figure 1 shows the sample distribution of the derived kernel K_m in an m-layer architecture, for $1 \le m \le 4$.[b] Each plot shows the histogram of $K_m(f,g)$ computed using every pair of different f, g from a collection of 500 images randomly sampled from the

[b]Details of implementation: we use the normalized inner product kernel and max pooling function. For each $1 \le m \le 4$ we choose the patch sizes of the m-layer architecture to be uniformly spaced up to 28×28 pixels. The initial feature map is

MNIST dataset of handwritten digits [32]. In particular, the histogram for the 1-layer architecture corresponds to the distribution of the normalized inner product between the original images. Note the different scales on the plots. The horizontal axis in each plot shows the empirical range of the kernel values.

From Figure 1 we clearly see the effect of range compression. The empirical lower bound of the derived kernel is drastically increased at each layer. In particular, in the case of the 4-layer architecture we see that the effective range of the derived kernel has a width of approximately 10^{-8}. Since single precision variables in MATLAB can represent values up to about seven decimal places, Figure 1 suggests that the kernel values in a 4-layer architecture would be indistinguishable in MATLAB with single precision floating point computations. Indeed, this phenomenon is reflected in the poor classification performance of a 4-layer architecture, as we shall see in Section 3.4.

3.1 Setting and Preliminaries

We first introduce some definitions and notations that we are using in this section. Throughout, let $\mathbb{R}_0$ denote the set of nonnegative real numbers and ℓ_0^2 denote the set of square summable nonnegative sequences,

$$\ell_0^2 = \{\alpha = (\alpha_i)_{i\in\mathbb{N}} \in \ell^2 \;\mid\; \alpha_i \geq 0 \;\text{ for }\; i \in \mathbb{N}\} \;.$$

Given $x \in \ell^2 \setminus \{0\}$, let $\widehat{x} = x/\|x\|$. Similarly, given $m \in \mathbb{N}$, let $\widehat{N}_m$ denote the normalized neural response at layer m,

$$\widehat{N}_m(f) = \frac{N_m(f)}{\|N_m(f)\|} \quad \text{for } f \in \mathrm{Im}(v_m) \;,$$

and let $\widehat{K}_m$ denote the normalized inner product between the neural responses at layer m,

$$\widehat{K}_m(f,g) = \left\langle \widehat{N}_m(f),\, \widehat{N}_m(g) \right\rangle \quad \text{for } f, g \in \mathrm{Im}(v_m) \;.$$

3.1.1 Nonnegative Architectures

In the analysis of range compression we shall only consider a *nonnegative architecture*, which is a hierarchical architecture where the neural response and derived kernel at each layer only take nonnegative values. Specifically, to obtain a nonnegative architecture it suffices to make the following assumptions:

- **Initial feature map.** The initial feature map N_1 maps the functions in $\mathrm{Im}(v_1)$ to vectors of nonnegative values, that is,

  $$N_1 \colon \mathrm{Im}(v_1) \to \ell_0^2 \;.$$

  For example, in the case of images, we can take N_1 to be the vector representation of the pixel values of the images.

- **Kernel function.** The kernel function $K \colon \ell^2 \times \ell^2 \to \mathbb{R}$ maps nonnegative inputs to nonnegative outputs, that is,

  $$K(x,y) \geq 0 \quad \text{if } x, y \in \ell_0^2 \;.$$

  For example, the inner product, normalized inner product, and Gaussian kernels satisfy this nonnegativity property.

- **Pooling function.** The pooling function Ψ acts on nonnegative inputs and produces nonnegative outputs, that is,

  $$\Psi \colon \mathbb{R}_0^* \to \mathbb{R}_0 \;,$$

the vector representation of the pixel values. At each layer we take the template set to be a set of 500 templates of the first kind, randomly sampled from the MNIST dataset. Finally, we choose the transformations at each layer to be all possible translations from one patch to the next larger patch.

where $\mathbb{R}_0^* = \bigcup_{n\in\mathbb{N}} \mathbb{R}_0$ is the set of all finite-length sequences of nonnegative real numbers. For example, we can restrict the domains of the pooling functions given in Table 2 to satisfy this requirement.

- **Templates.** Each template set $\mathcal{T}_m$ consists of nonnegative linear combinations of templates of the first kind.

In particular, we will take the template set $\mathcal{T}_m$ to be a finite set of convex combinations of templates of the first kind. That is, if $\Phi\colon \ell^2 \to \ell^2$ is the feature map corresponding to the kernel function $K\colon \ell^2 \times \ell^2 \to \mathbb{R}$, then every template $\tau \in \mathcal{T}_m$ is of the form

$$\tau = \sum_{i=1}^{d} c_i\, \Phi(N_m(f_i)) \quad \text{for some } \; d \in \mathbb{N},\; f_i \in \mathrm{Im}(v_m), \; \text{and } \; c_i \geq 0 \; \text{ with } \; \sum_{i=1}^{d} c_i = 1 \;\; .$$

For brevity, we will refer to this set of templates as *convex templates*. Note that templates of the first kind are convex templates.

3.1.2 Strongly Bounded Pooling Functions

In addition to the assumption regarding the nonnegativity of the pooling function Ψ, we will also assume that Ψ satisfies the following property, which states that Ψ maps bounded inputs to bounded outputs.

Definition 3.1 (Strongly bounded pooling function)**.** *A pooling function $\Psi\colon \mathbb{R}_0^* \to \mathbb{R}_0$ is strongly bounded if there exists a non-decreasing concave function $\zeta\colon \mathbb{R}_0 \to \mathbb{R}_0$ with the property that for all $n \in \mathbb{N}$, we can find $C_n > 0$ such that*

$$C_n\, \zeta\Big(\min_{1\leq i\leq n} \alpha_i\Big) \leq \Psi(\alpha) \leq C_n\, \zeta\Big(\max_{1\leq i\leq n} \alpha_i\Big) \quad \text{for all } \alpha = (\alpha_1, \ldots, \alpha_n) \in \mathbb{R}_0^n \;\; .$$

In this case, we also say that Ψ is a strongly bounded pooling function that is dominated by ζ.

Strong boundedness is a relatively mild condition and is satisfied by most pooling functions that we consider in practice. For example, all the pooling functions given in Table 2 are strongly bounded with ζ being the identity function, $\zeta(x) = x$ for $x \in \mathbb{R}_0$. Specifically, the average, max, and ℓ^∞-norm pooling functions are strongly bounded with the constant $C_n = 1$. On the other hand, the ℓ^1-norm pooling function is strongly bounded with the constant $C_n = n$.

We use the term *strongly bounded* because when the dominating function ζ is the identity function (which is the case for most common pooling functions, as we saw in the preceding paragraph), a strongly bounded pooling function is also weakly bounded. However, in general there is no relation between strong and weak boundedness. For example, a constant nonzero pooling function is not weakly bounded, but it is strongly bounded with ζ being a constant function.

Evidently the zero pooling function is not very interesting since, for example, in this case the neural response is identically zero. Thus in the remainder of this section we impose an additional assumption that we only consider a strongly bounded pooling function that is not the zero pooling function. Equivalently, we assume that the dominating function ζ satisfies $\zeta(x) > 0$ for $x > 0$.

Finally, noting that the composition of two non-decreasing concave functions is again non-decreasing and concave, we have the following simple result, which is particularly important in Section 3.3 when we are considering normalization procedures in the pooling function.

Lemma 3.2. *If $\Psi\colon \mathbb{R}_0^* \to \mathbb{R}_0$ is a strongly bounded pooling function and $\sigma\colon \mathbb{R}_0 \to \mathbb{R}_0$ is a non-decreasing concave function, then $\Psi \circ \sigma$ is also a strongly bounded pooling function.*

3.1.3 Dynamic Range of the Derived Kernel

Since the values of the derived kernel can have different scales from layer to layer, in order to compare the properties of the derived kernel at different layers we need a standardized measure. The following concept of dynamic range achieves this goal by measuring the spread of the derived kernel relative to its own absolute scale.

Definition 3.3 (Dynamic range)**.** *For each $m \in \mathbb{N}$, the dynamic range of the derived kernel K_m is the ratio between the smallest and largest values of K_m,*

$$\delta(K_m) = \frac{\inf_{f,g \in Im(v_m)} K_m(f,g)}{\sup_{f,g \in Im(v_m)} K_m(f,g)} \ .$$

Because we only consider architectures with nonnegative values, we see that $0 \leq \delta(K_m) \leq 1$ for $m \in \mathbb{N}$.[c] A larger $\delta(K_m)$ corresponds to a "smaller" range of values in the sense that the values of K_m are more tightly concentrated, even though their absolute values can be arbitrarily large. We will state the range compression results in terms of the dynamic range of the derived kernel.

More precisely, we will show that under the presence of normalization procedures, the dynamic range $\delta(K_m)$ converges to 1 as $m \to \infty$. As stated earlier, our approach is to show that $\delta(K_m)$ is an increasing function of m. The following theorem then implies that we also have the convergence of the normalized neural responses.

Theorem 3.4. *Consider a nonnegative architecture with a strongly bounded pooling function and convex templates. If the dynamic range of the derived kernel converges to its maximum value,*

$$\lim_{m \to \infty} \delta(K_m) = 1 \ ,$$

then the normalized neural response converges to a constant vector,

$$\lim_{m \to \infty} \sup_{f,g \in Im(v_m)} \|\widehat{N}_m(f) - \widehat{N}_m(g)\| = 0 \ .$$

We note that the convergence result in Theorem 3.4 is expressed only in terms of the normalized neural response because we do not make any assumptions on the size of the template sets. In general, the ℓ^2 distance $\|N_m(f) - N_m(g)\|$ will depend on the dimension of the space in which the neural responses $N_m(f)$ lie, which is $|\mathcal{T}_{m-1}|$. Therefore, even when the componentwise differences between $N_m(f)$ and $N_m(g)$ are small, we can make $\|N_m(f) - N_m(g)\|$ arbitrarily large by increasing the cardinality of $\mathcal{T}_{m-1}$.

Remark 3.5 (Range compression in architectures with extension)**.** Consider the extension step described in Remark 2.9, and recall from (2) that the derived kernel K_{w_m} at the extension step can be expressed as a summation of the derived kernel K_{v_m} at the previous layer. From (2) we immediately see that the dynamic range of K_{w_m} is bounded below by the dynamic range of K_{v_m},

$$\delta(K_{w_m}) \geq \delta(K_{v_m}) \ .$$

This means if an architecture suffers from range compression, then when we extend the architecture by inserting extension steps, the extended architecture also suffers from range compression.

3.2 Range Compression: Normalized Kernel Functions

The first approach to perform normalization is to use a normalized kernel function. Recall from Section 2.1 that a kernel $K \colon \ell^2 \times \ell^2 \to \mathbb{R}$ is *normalized* if $K(x,x) = 1$ for all $x \in \ell^2$. In particular, by the Cauchy–Schwarz inequality this implies $K(x,y) \leq 1$ for all $x, y \in \ell^2$. Moreover, since at every layer $m \in \mathbb{N}$ the derived kernel K_m achieves its maximum value,

$$\sup_{f,g \in \mathrm{Im}(v_m)} K_m(f,g) = K_m(f,f) = 1 \ ,$$

[c] Note that here we assume $\sup_{f,g \in \mathrm{Im}(v_m)} K_m(f,g) > 0$, for otherwise $K_m(f,g) = 0$ for all $f, g \in \mathrm{Im}(v_m)$.

in this case the dynamic range of the derived kernel is equal to its minimum value,

$$\delta(K_m) = \inf_{f,g \in \mathrm{Im}(v_m)} K_m(f,g) \ .$$

The range compression result in the case of normalized kernel is the following.

Theorem 3.6. *Consider a nonnegative architecture with a normalized kernel K, a strongly bounded pooling function Ψ, and convex templates. Assume that the feature map Φ corresponding to K is 1-Lipschitz with respect to the normalized metric,*

$$\|\Phi(x) - \Phi(y)\| \leq \|\widehat{x} - \widehat{y}\| \quad \text{for all } x, y \in \ell^2 \setminus \{0\} \ . \tag{3}$$

Then at every layer $m \in \mathbb{N}$,

$$\delta(K_{m+1}) \geq \frac{2\delta(K_m)}{1 + \delta(K_m)^2} \ . \tag{4}$$

In particular, if at some layer $m \in \mathbb{N}$ we have $\delta(K_m) > 0$, then

$$\lim_{n \to \infty} \delta(K_n) = 1 \ .$$

Now we consider examples of normalized kernel functions that satisfy the hypothesis of Theorem 3.6.

Example 3.7. The normalized inner product kernel

$$K(x,y) = \frac{\langle x, y \rangle}{\|x\| \, \|y\|}$$

has the associated feature map $\Phi(x) = \widehat{x}$, which satisfies the Lipschitz continuity condition in (3) with equality.

Example 3.8. The Gaussian kernel with the normalized inner product

$$K(x,y) = e^{-\gamma \|\widehat{x} - \widehat{y}\|^2} = e^{-2\gamma(1 - \langle \widehat{x}, \widehat{y} \rangle)}$$

is 1-Lipschitz when $0 < \gamma \leq 1/2$. This can be seen as follows. Since the Gaussian kernel is normalized, condition (3) is equivalent to requiring $K(x,y) \geq \langle \widehat{x}, \widehat{y} \rangle$ for all $x, y \in \ell^2 \setminus \{0\}$. This inequality is clearly true when $\langle \widehat{x}, \widehat{y} \rangle \leq 0$ since the Gaussian kernel is nonnegative. Thus, considering only the case when $a = \langle \widehat{x}, \widehat{y} \rangle \geq 0$, we then must have

$$\gamma \leq \inf_{0 \leq a \leq 1} \frac{\log(1/a)}{2(1-a)} = \frac{1}{2} \ .$$

The value $1/2$ is obtained by noting that the function that we are trying to minimize above is decreasing in a, so its minimum occurs at $a = 1$, and indeed it is easy to show that the minimum value is $1/2$ by an application of the L'Hôspital's rule.

3.3 Range Compression: Normalized Pooling Functions

Another approach to perform normalization is to use a pooling function that has a thresholding effect. Namely, the pooling function treats the input values as if they lie in a bounded region by clipping the values that are higher than some threshold. This thresholding behavior is usually achieved by choosing a pooling function has a horizontal asymptote, for example by considering a pooling function of the form

$$\widetilde{\Psi} = \Psi \circ \sigma \ ,$$

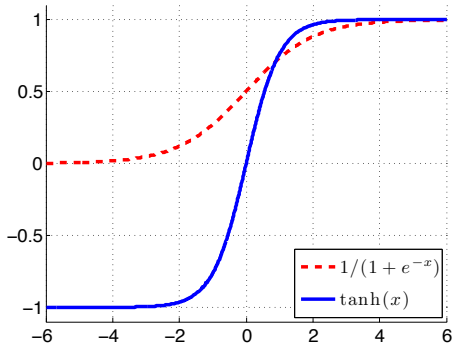


Figure 2: Plots of the sigmoid functions $\sigma(x) = \frac{1}{1+e^{-x}}$ and $\sigma(x) = \tanh(x)$.

where Ψ is a pooling function and σ is a sigmoid function. Some sigmoid functions that are commonly used in practice are the logistic function $\sigma(x) = \frac{1}{1+e^{-x}}$ and the hyperbolic tangent $\sigma(x) = \tanh x$ (see Figure 2).

When we are working with a nonnegative architecture, the presence of the sigmoid function also leads to range compression. Intuitively, this is because the sigmoid function is approximately constant for sufficiently large inputs, which implies that the values of the derived kernel converge to a stable limit. Accordingly, a sigmoid function that approaches its asymptote more slowly should be more resistant to range compression than a sigmoid function that rises sharply. Indeed this is the case, as we shall see in Example 3.13.

Before proceeding further with our discussion, we first need to formalize the notion of a sigmoid function. As we are working with a nonnegative architecture, we only need to define a sigmoid function σ on $\mathbb{R}_0$. However, for the sake of completeness we can also define σ on $(-\infty, 0)$ by assuming that σ is antisymmetric with respect to 0,

$$\sigma(-x) = 2\sigma(0) - \sigma(x) \quad \text{for all } x \in \mathbb{R}_0 \ .$$

For example, the sigmoid functions shown in Figure 2 satisfy this property.

Definition 3.9 (Sigmoid)**.** *A sigmoid function is a non-decreasing, concave, and differentiable function* $\sigma\colon \mathbb{R}_0 \to \mathbb{R}_0$ *with a horizontal asymptote,*

$$\lim_{x\to\infty} \sigma(x) < \infty \ .$$

We note that the assumption that σ is concave and has a horizontal asymptote actually implies that σ is non-decreasing on $\mathbb{R}_0$. Moreover, as the case when σ is the constant zero function is of hardly any interest to us, we further assume that a sigmoid function σ satisfies $\sigma(x) > 0$ for $x > 0$. Next, recall from Lemma 3.2 that if Ψ is a strongly bounded pooling function then $\Psi \circ \sigma$ is also a strongly bounded pooling function. This is the form of the pooling function that we use in Theorem 3.10 below.

Now, given a sigmoid function σ, the assumption that σ is concave and has a horizontal asymptote implies that σ approaches its asymptote at a certain rate (see Lemma B.4 in Appendix B.3). In particular, we can show that

$$F_\sigma(x) := \frac{2x\sigma'(x)}{\sigma(x)} \to 0 \quad \text{as } x \to \infty \ .$$

Intuitively, this quantity $F_\sigma(x)$ measures how fast the sigmoid function converges to its asymptote. A sigmoid function σ that converges fast to its asymptote corresponds to a function F_σ that approaches 0 more quickly, which in turn will yield a higher rate of convergence of the derived kernel.

Specifically, given $0 < \eta < 1$, the fact that $F_\sigma(x) \to 0$ implies that $F_\sigma(x) \leq \eta$ for all sufficiently large x. Let $\varepsilon_{\eta,\sigma}$ denote the first time this happens,

$$\varepsilon_{\eta,\sigma} = \inf\left\{b \in \mathbb{R}_0 \mid \eta \geq F_\sigma(x) \text{ for all } x \geq b\right\} . \tag{5}$$

The quantity $\varepsilon_{\eta,\sigma}$ plays the role of limiting the effective region of range compression, in the sense that our result only applies if the values of the derived kernel are at least $\varepsilon_{\eta,\sigma}$. We shall see that for common pooling functions the value of $\varepsilon_{\eta,\sigma}$ is very small, so Theorem 3.10 is applicable in practice.

Theorem 3.10. *Consider a nonnegative architecture with the inner product kernel, convex templates, and a strongly bounded pooling function of the form*

$$\widetilde{\Psi} = \Psi \circ \sigma ,$$

where Ψ is a strongly bounded pooling function that is dominated by ζ and σ is a sigmoid function. Given $m \in \mathbb{N}$, if we have

$$\inf_{f,g \in Im(v_m)} K_m(f,g) \geq \varepsilon_{\eta,\sigma} \quad \text{for some } 0 < \eta < 1 , \tag{6}$$

then

$$\delta(K_{m+1}) \geq \delta(K_m)^\eta .$$

Moreover, if (6) holds at layer $m \in \mathbb{N}$ and

$$|\mathcal{T}_m| \geq \frac{\varepsilon_{\eta,\sigma}}{C_{|H_m|}^2 \, \zeta\big(\sigma(\varepsilon_{\eta,\sigma})\big)^2} , \tag{7}$$

then (6) also holds at layer $m+1$ with the same value of $0 < \eta < 1$.[d] In particular, if (6) holds at some layer $m \in \mathbb{N}$ with $\delta(K_m) > 0$ and (7) holds at all layers $n \geq m$, then

$$\lim_{n \to \infty} \delta(K_n) = 1 .$$

We consider some examples of sigmoid functions that are commonly used in practice and illustrate the values of the various parameters in Theorem 3.10 above.

Example 3.11 (Hyperbolic tangent)**.** Consider $\sigma(x) = \tanh(x)$. Then $\sigma'(x) = 1/\cosh(x)$, and

$$F_\sigma(x) = \frac{2x\sigma'(x)}{\sigma(x)} = \frac{2x}{\sinh(x)} .$$

Figure 3(a) shows the plot of the function $F_\sigma(x)$ for $\sigma(x) = \tanh(x)$. For $\eta = 0.5$, for example, we have $\varepsilon_{\eta,\sigma} \approx 3.26$. When the dominating function ζ is the identity function and the constant $C_{|H_m|}$ is 1 (for example, when Ψ is the max or average pooling function), the condition in (7) becomes

$$|\mathcal{T}_m| \geq \frac{\varepsilon_{\eta,\sigma}}{\sigma(\varepsilon_{\eta,\sigma})^2} \approx \frac{3.26}{\tanh^2(3.26)} \approx 3.28 .$$

Example 3.12 (Logistic function)**.** Now consider the logistic function $\sigma(x) = \frac{1}{1+e^{-x}}$. Then

$$\sigma'(x) = \sigma(x)\big(1 - \sigma(x)\big) = \frac{e^{-x}}{(1+e^{-x})^2} ,$$

and

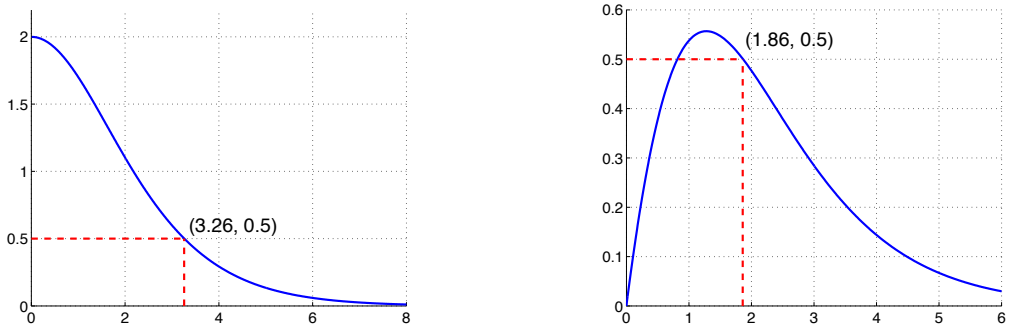
$$F_\sigma(x) = \frac{2x\sigma'(x)}{\sigma(x)} = 2x\big(1 - \sigma(x)\big) = \frac{2xe^{-x}}{1+e^{-x}} .$$

[d] Here $C_{|H_m|}$ is the constant in the definition of the strong boundedness of Ψ in Definition 3.1.

Figure 3(b) shows the plot of the function $F_\sigma(x)$ for $\sigma(x) = \frac{1}{1+e^{-x}}$. For $\eta = 0.5$ we have $\varepsilon_{\eta,\sigma} \approx 1.86$. When the dominating function ζ is the identity function and $C_{|H_m|} = 1$, the condition in (7) becomes

$$|\mathcal{T}_m| \geq \frac{\varepsilon_{\eta,\sigma}}{\sigma(\varepsilon_{\eta,\sigma})^2} \approx 1.86\,(1 + e^{-1.86})^2 \approx 2.48 \ .$$

Interestingly, we see from Figure 3(b) that when σ is the logistic function we actually have $F_\sigma(x) < 0.6$ for all $x \in \mathbb{R}_0$. This means $\varepsilon_{\eta,\sigma} = 0$ for $\eta = 0.6$, and hence the convergence assertion in Theorem 3.10 holds without further assumptions on the template sets $\mathcal{T}_m$.



(a) The function $F_\sigma(x)$ for $\sigma(x) = \tanh(x)$.

(b) The function $F_\sigma(x)$ for $\sigma(x) = \frac{1}{1+e^{-x}}$.

Figure 3: Plots of the function $F_\sigma(x) = \frac{2x\sigma'(x)}{\sigma(x)}$ for some common choices of the sigmoid function σ.

Example 3.13 (Slow sigmoid function)**.** Finally, consider a sigmoid function that approaches its asymptote slowly. For example, given $k \geq 1$ and a sigmoid function σ, we can take $\sigma_k(x) = \sigma(x/k)$. The function σ_k is obtained by stretching the domain of σ by a factor of k, so that σ_k approaches its asymptote k times slower than σ. In this case we have $\sigma_k'(x) = \frac{1}{k}\sigma'(x/k)$, and

$$F_{\sigma_k}(x) = \frac{2x\sigma_k'(x)}{\sigma_k(x)} = \frac{2x\sigma'(x/k)}{k\sigma(x/k)} = F_\sigma\left(\frac{x}{k}\right) \ .$$

Therefore, from the definition of $\varepsilon_{\eta,\sigma}$ in (5) we see that

$$\varepsilon_{\eta,\sigma_k} = k\varepsilon_{\eta,\sigma} \ .$$

That is, a slower sigmoid function imposes a stronger hypothesis on Theorem 3.10, namely the lower bound of K_m in (6) becomes larger. We can also estimate the lower bound on the number of templates in (7). Given $0 < \eta < 1$ and $m \in \mathbb{N}$, let $M_\sigma \equiv M_{m,\eta,\sigma}$ denote the lower bound on the number of templates defined by (7),

$$M_\sigma = \frac{\varepsilon_{\eta,\sigma}}{C_{|H_m|}^2\,\zeta\big(\sigma(\varepsilon_{\eta,\sigma})\big)^2} \ .$$

When we use σ_k as the sigmoid function, this lower bound becomes

$$M_{\sigma_k} = \frac{\varepsilon_{\eta,\sigma_k}}{C_{|H_m|}^2\,\zeta\big(\sigma(\varepsilon_{\eta,\sigma_k})\big)^2} = \frac{k\varepsilon_{\eta,\sigma}}{C_{|H_m|}^2\,\zeta\big(\sigma(k\varepsilon_{\eta,\sigma})\big)^2} \leq \frac{k\varepsilon_{\eta,\sigma}}{C_{|H_m|}^2\,\zeta\big(\sigma(\varepsilon_{\eta,\sigma})\big)^2} = kM_\sigma \ .$$

This means when we use σ_k as the sigmoid function, condition (7) in Theorem 3.10 can be replaced with

$$|\mathcal{T}_m| \geq \frac{k\varepsilon_{\eta,\sigma}}{C_{|H_m|}^2 \, \zeta\big(\sigma(\varepsilon_{\eta,\sigma})\big)^2}$$

to guarantee that (6) still holds at subsequent layers.

3.4 Empirical Results and Possible Fixes to Range Compression

We now present some empirical results related to range compression and propose several possible ways to alleviate this issue. In this section we focus on the case of the normalized inner product kernel (Example 3.7), but our discussion can be extended to the general setting in a straightforward manner.

We saw in Figure 1 that the rate at which range compression occurs in practice is much faster than the lower bound given in Theorem 3.6. In particular, Figure 1 suggests that single precision floating points cannot distinguish the derived kernel values in a 4-layer architecture. Indeed, this observation is reflected in the poor performance of a 4-layer architecture when we are working with single precision variables in MATLAB, as shown in Figure 4. We measure the performance using the classification accuracy on the image classification task on the MNIST dataset. The classification rule is 1-nearest neighbor using the derived kernel as the similarity measure.[e]

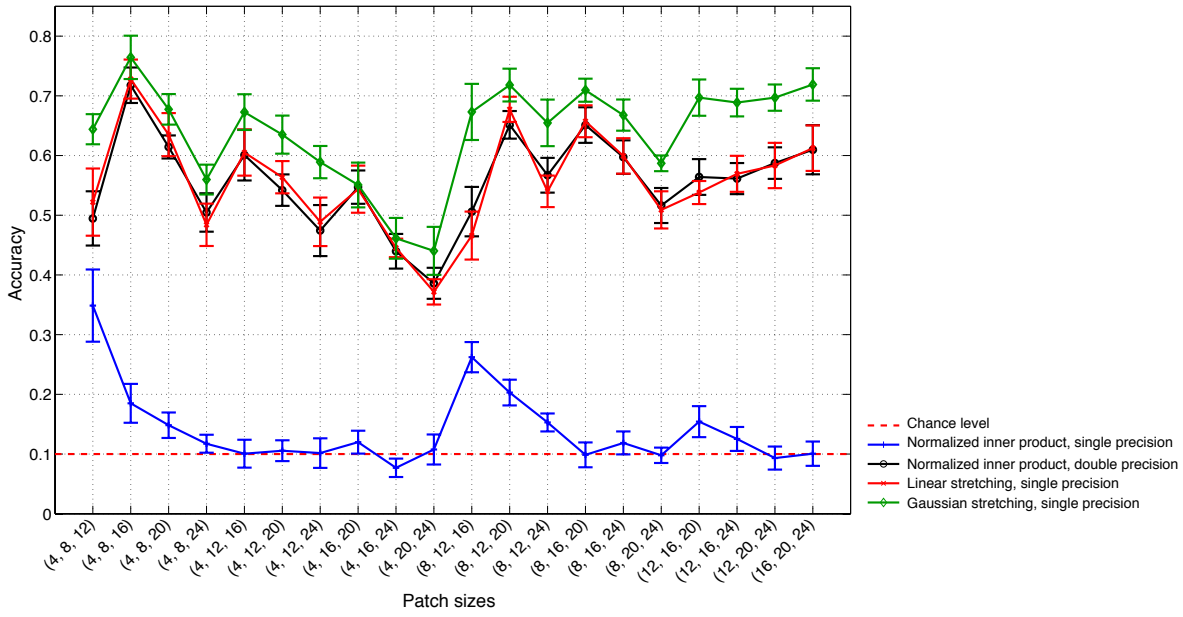


Figure 4: Average accuracy on the MNIST digit classification task using a 4-layer architecture. The performance of the normalized inner product kernel with single precision is significantly lower than that with double precision due to range compression. The proposed stretching techniques can restore the accuracy of the single precision to be comparable with the accuracy of the double precision.

[e]Details of implementation: we perform each experiment in a 4-layer architecture using the normalized inner product kernel and max pooling function. We fix the largest patch size to 28×28 pixels and vary the smaller patch sizes. The initial feature map is the vector representation of the pixel values. At each layer we use 500 randomly sampled templates of the first kind and all possible translations. Both the training and testing sets consist of 30 images per digit class. The classification rule is 1-nearest neighbor using the derived kernel as the similarity measure. Each reported accuracy is an average over 10 independent trials.

From Figure 4 we see that the accuracy of the normalized inner product kernel with single precision is significantly lower than that with double precision. In fact, the average accuracy of the normalized inner product kernel with single precision is very close to the chance level $1/10 = 0.1$. This confirms our observation that range compression on a 4-layer architecture is severe enough that the derived kernel values are indistinguishable using single-precision floating points. Moreover, this result shows that range compression can render a multilayer architecture unusable due to the saturation effect.

There are several possible ways to counteract the effect of range compression, based on the idea of introducing a tuning parameter to stretch the dynamic range of the derived kernel. Namely, given a derived kernel K_m that suffers from range compression, we build a *stretched derived kernel* $\widetilde{K}_m$ that recovers the dynamic range of K_m. The stretched derived kernel $\widetilde{K}_m$ is then used to construct the neural response N_{m+1} at the next layer, following Definition 2.6. Since we showed that range compression occurs layerwise, the derived kernel K_{m+1} will again be subject to range compression, and we can repeat the stretching technique at each subsequent layer.

The stretching step from K_m to $\widetilde{K}_m$ can be interpreted as adapting each layer of the architecture to work on the effective range of the kernel function. This is in part inspired by the evidence that neurons in the brain only represent information with a finite amount of bits [49]. Therefore, if range compression or a similar effect occurs in the brain, then the each neuron will have to learn to operate in the effective range of the data that it is working with.

We now describe two possible stretching techniques and see how they affect the performance of the architecture.

- **Linear stretching.** Given the derived kernel K_m, choose the stretched derived kernel $\widetilde{K}_m$ to be a linear transformation that stretches the range of K_m to be $[0, 1]$. More specifically, let

$$a = \inf_{f,g \in \mathrm{Im}(v_m)} K_m(f, g) \ ,$$

  so that $a \leq K_m(f, g) \leq 1$ for all $f, g \in \mathrm{Im}(v_m)$. Then by choosing

$$\widetilde{K}_m(f, g) = \frac{K_m(f, g) - a}{1 - a} \quad \text{for all } f, g \in \mathrm{Im}(v_m) \ ,$$

  we ensure that

$$\inf_{f,g \in \mathrm{Im}(v_m)} \widetilde{K}_m(f, g) = 0 \quad \text{and} \quad \sup_{f,g \in \mathrm{Im}(v_m)} \widetilde{K}_m(f, g) = 1 \ .$$

- **Gaussian stretching.** Another stretching strategy is to consider a Gaussian tuning operation at each layer. More precisely, we take the stretched derived kernel to be

$$\widetilde{K}_m(f, g) = e^{-\gamma_m \|\widehat{N}_m(f) - \widehat{N}_m(g)\|^2} = e^{-2\gamma_m(1 - K_m(f,g))} \quad \text{for all } f, g \in \mathrm{Im}(v_m) \ ,$$

  where $\gamma_m \geq 0$ is a scale parameter that can vary from layer to layer. Intuitively, even if $K_m(f, g)$ is very close to 1, we can choose γ_m big enough to spread the range of $\widetilde{K}_m(f, g)$. Contrast this with the case of Gaussian tuning with a fixed scale in Example 3.8, in which case we still have range compression.

  For example, given the stretched derived kernel $\widetilde{K}_{m-1}$ at layer $m - 1$, we can choose γ_m such that

$$\inf_{f,g \in \mathrm{Im}(v_m)} \widetilde{K}_m(f, g) = \inf_{f,g \in \mathrm{Im}(v_{m-1})} \widetilde{K}_{m-1}(f, g) \ ,$$

  so that the range of the stretched derived kernel $\widetilde{K}_m(f, g)$ at each layer remains the same. Letting

$$a = \inf_{f,g \in \mathrm{Im}(v_m)} K_m(f, g) \quad \text{and} \quad b = \inf_{f,g \in \mathrm{Im}(v_{m-1})} \widetilde{K}_{m-1}(f, g) \ ,$$

then we want to choose γ_m such that

$$\inf_{f,g \in \mathrm{Im}(v_m)} \widetilde{K}_m(f,g) = \inf_{f,g \in \mathrm{Im}(v_m)} e^{-2\gamma_m(1-K_m(f,g))} = e^{-2\gamma_m(1-a)} = b \ ,$$

which gives us

$$\gamma_m = -\frac{\log b}{2(1-a)} \ .$$

We check empirically that these stretching techniques restore the dynamic range of the derived kernel by computing the sample distribution of $K_m(f,g)$ in an m-layer architecture, for $1 \leq m \leq 4$, following the same procedure as in Figure 1. Figure 5 shows the resulting histograms in the case of the normalized inner product kernel (no stretching), linear stretching, and Gaussian stretching. Evidently both the linear and Gaussian stretching techniques succeed in restoring the dynamic range of the derived kernel. However, we observe that since linear stretching assigns equal weight on each interval of values in the transformation, the resulting distribution tends to be skewed to the right with a long tail. On the other hand, Gaussian stretching manages to recover the symmetry of the distribution better.

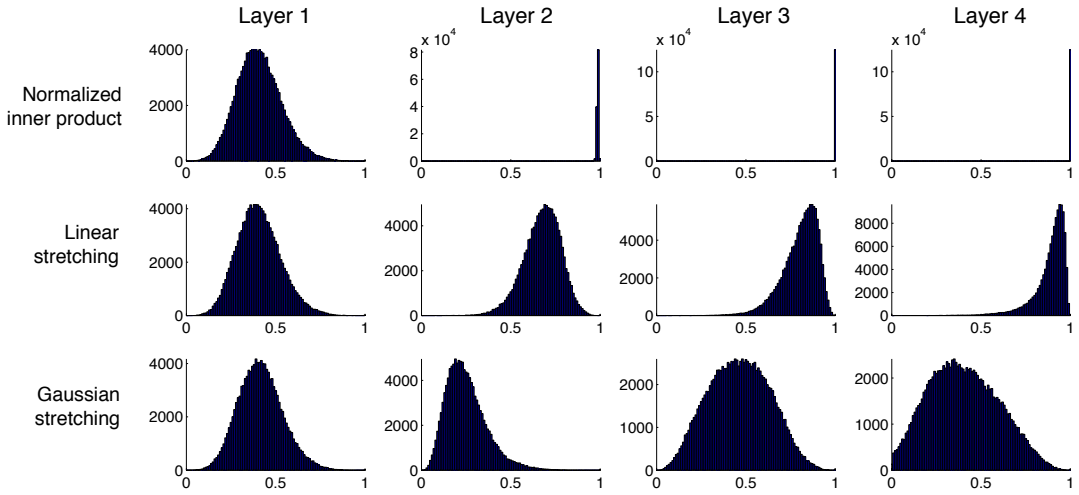


Figure 5: Sample distributions of $K_m(f,g)$ in an m-layer architecture, for $1 \leq m \leq 4$, using the normalized inner product kernel, linear stretching, and Gaussian stretching. Both stretching techniques restore the dynamic range of the derived kernel, but Gaussian stretching recovers a more symmetric distribution.

Furthermore, our experiments also confirm that both linear and Gaussian stretching techniques can restore the performance of the 4-layer architecture in the single precision environment. Figure 4 shows that the classification accuracy of the linear stretching with single precision is virtually identical to the accuracy of the normalized inner product kernel with double precision. Moreover, the accuracy of the Gaussian stretching with single precision is slightly higher compared to that of the linear stretching, which might be attributed to the earlier observation that Gaussian stretching recovers a more symmetric distribution.

These results, in particular the performance of the normalized inner product kernel with double precision, suggest that range compression by itself does not lead to poor performance as long as we are working in an environment with sufficiently fine precision. However, range compression is prevalent in many algorithms that perform normalization, and consequently, range compression prohibits us from taking advantage of the faster processing speed of GPUs, which commonly use single precision floating point computations.

3.5 Discussion

We have seen that range compression is a ubiquitous property that presents a serious problem for experiments and empirical studies of hierarchical architectures. However, our analysis currently relies on the assumption that we are working with a nonnegative architecture. It remains to be seen if range compression also occurs when we allow the architecture to use negative values.

Glorot and Bengio [18] reported that they observed saturated hidden units in the feedforward neural networks (that can have positive and negative values) when they used the logistic function, hyperbolic tangent, and softsign function $\sigma(x) = x/(1 + |x|)$ [6] as sigmoid functions. Moreover, Glorot and Bengio also observed that the saturation effect in the case of the softsign function is less severe and the performance of the model is better than the case of the logistic and hyperbolic tangent functions. Since the softsign function has a polynomial convergence rate to its asymptote, as opposed to exponential rate in the case of the logistic and hyperbolic tangent functions, this latter observation is in accordance to our remark in Section 3.3 that a slower sigmoid function is more robust to range compression.

However, in order to fully explain the saturation effect that Glorot and Bengio reported, we need to extend our theory of range compression to the general case when the architecture can take positive and negative values. We suspect that the range compression and convergence results in this case will be probabilistic. The first step that we can take is to study the case when the negative values are introduced by the normalization procedure.

For example, a normalization method that is suggested in practice [45] is to take the derived kernel to be the normalized cross correlation of the neural responses. Specifically, suppose that for some $m \geq 2$ the template set $\mathcal{T}_{m-1}$ is finite. Given the neural response N_m at layer m, we can consider the *centered neural response* $N_m^c \colon \mathrm{Im}(v_m) \to \ell^2(\mathcal{T}_{m-1})$ given by

$$N_m^c(f)(\tau) = N_m(f)(\tau) - \frac{1}{|\mathcal{T}_{m-1}|} \sum_{\tau' \in \mathcal{T}_{m-1}} N_m(f)(\tau') \quad \text{for all } f \in \mathrm{Im}(v_m) \text{ and } \tau \in \mathcal{T}_{m-1} \ .$$

The derived kernel K_m is then given by the normalized inner product of the centered neural responses,

$$K_m(f, g) = \frac{\langle N_m^c(f),\ N_m^c(g) \rangle}{\|N_m^c(f)\|\ \|N_m^c(g)\|} \quad \text{for all } f, g \in \mathrm{Im}(v_m) \ .$$

Note that some of the components of the centered neural responses will be negative, so the derived kernel values can in general be negative.

On the other hand, Jarrett et al. [26] recently emphasized the importance of positivity in the architecture to improve performance, typically by rectifying the values before the pooling step. In our framework, this corresponds to using a pooling function that is derived from a norm, for example the ℓ^1-norm or ℓ^∞-norm pooling function. The brief presence of negative values at each layer might contribute to counteracting range compression, achieving a similar effect to the stretching techniques that we described in Section 3.4.

4 Linear Architectures

An oft-emphasized point about hierarchical algorithms is the importance of nonlinearities in the architecture. Several authors argued that nonlinearities are necessary to represent complex intelligent behaviors [3, 18], and indeed nonlinearities have generally been considered as a prerequisite for hierarchical architectures [5]. Furthermore, the max pooling function has been consistently shown to outperform the average pooling function in classification tasks, especially when the remainder of the architecture is linear [7, 26, 62]. Boureau et al. [7] proposed an explanation that the max pooling function has a better performance because it is more robust to clutter than the average pooling function. Another argument that might explain this contrast is to recall that one of the motivation for building a hierarchical architecture is to achieve a level of complexity that cannot be captured by shallow architectures, and the presence of

nonlinearities is essential to prevent the architecture from collapsing into a one-stage linear computation. In this section we investigate this issue and study the properties and limitations of linear architectures using the framework of the derived kernel.

We first formally define the notion of a linear architecture. We say that a pooling function $\Psi\colon \mathbb{R}^* \to \mathbb{R}$ is *linear* if each restriction of Ψ on inputs of a certain length is linear. That is, Ψ is linear if for each $n \in \mathbb{N}$ we have

$$\Psi(\alpha + \beta) = \Psi(\alpha) + \Psi(\beta) \quad \text{and} \quad \Psi(c\alpha) = c\Psi(\alpha) \quad \text{for every } \alpha, \beta \in \mathbb{R}^n,\ c \in \mathbb{R} \ .$$

Equivalently, Ψ is linear if for each $n \in \mathbb{N}$ we can find $\omega \in \mathbb{R}^n$ such that

$$\Psi(\alpha) = \langle \alpha,\, \omega \rangle \quad \text{for all } \alpha \in \mathbb{R}^n \ .$$

For example, the average pooling function is linear. A *linear architecture* is a hierarchical architecture with the normalized inner product kernel and a linear pooling function.[f] For $1 \leq m \leq n$, let $\mathcal{N}_m$ denote the closed linear span of the neural responses at layer m,

$$\mathcal{N}_m = \overline{\mathrm{Span}}\big(\{N_m(f) \mid f \in \mathrm{Im}(v_m)\}\big) \subseteq \ell^2 \ ,$$

and let $\dim(\mathcal{N}_m)$ denote the dimension of $\mathcal{N}_m$. Finally, let $\widehat{N}_m(f)$ denote the normalized neural response of a function $f \in \mathrm{Im}(v_m)$,

$$\widehat{N}_m(f) = \frac{N_m(f)}{\|N_m(f)\|} \ .$$

4.1 Rank Constraint

We first investigate how the choice of the parameters at the first layer limits the complexity of the architecture. We find that in a linear architecture, the dimensionality of the neural response at each layer is restricted by that of the first layer. We refer to this phenomenon as the *rank constraint*, stemming from the observation that when the template set $\mathcal{T}_m$ at each layer is finite and we conjoin the templates in $\mathcal{T}_m$ into a matrix, then the rank of the matrix is constrained by the rank of the templates matrix at the first layer.

Note that, however, this result on rank constraint holds without any assumption on the set of templates. In particular, this result means that when we use a linear architecture, using a large number of templates at each layer will result in an inefficient and redundant computation since the architecture is incapable of generating more complexity than that prescribed by the first layer.

Theorem 4.1. *Consider a linear architecture with $n \in \mathbb{N}$ layers. For each $1 \leq m \leq n-1$, we have*

$$\dim(\mathcal{N}_{m+1}) \leq \dim(\mathcal{N}_m) \ .$$

For example, in the case of images in computers, suppose at the first layer we are working with image patches of size $k \times k$ pixels. Suppose further that the initial feature map N_1 represents every such image patch as a vector of pixel intensities in $\mathbb{R}^{k^2}$. This means the space of neural responses $\mathcal{N}_1$ at the first layer has dimension at most k^2. In the case of a linear architecture, Theorem 4.1 then tells us that all subsequent spaces of neural responses $\mathcal{N}_m$ are at most k^2-dimensional, for $2 \leq m \leq n$, regardless of the number of templates that we are using.

4.2 Basis Independence

We now consider the case when we use an orthonormal basis of $\mathcal{N}_m$ as templates of the second kind $\mathcal{T}_m$ at each layer $1 \leq m \leq n-1$. This case is motivated by the use of several basis approximation techniques

[f] We can also use the inner product kernel, and all the results in this section still hold.

for templates learning in practice. For example, given a finite set of templates of the first kind at a layer, we can employ reduction schemes such as PCA or ICA to infer an approximate basis of the space of the neural responses $\mathcal{N}_m$. It is then reasonable to study the limiting case when we have a complete information about $\mathcal{N}_m$, in which case we are able to use an orthonormal basis of $\mathcal{N}_m$ as templates of the second kind.

Perhaps not surprisingly, the linearity of the pooling function allows the orthonormal basis of the neural responses to disappear from the final expression of the derived kernel. Just as the normalized inner product in a vector space is independent of the basis of the space, the derived kernel K_{m+1} is independent of the choice of the orthonormal basis of $\mathcal{N}_m$ that we use as templates of the second kind $\mathcal{T}_m$.

Theorem 4.2. *Consider a linear architecture with $n \in \mathbb{N}$ layers. Suppose that at each layer $1 \leq m \leq n-1$ the template set $\mathcal{T}_m$ is taken to be an orthonormal basis of $\mathcal{N}_m$. Then derived kernel K_{m+1} is independent of the choice of the basis of $\mathcal{N}_m$.*

We note that the proof of Theorem 4.2 above (provided in Appendix C.2) also shows that when we use the inner product kernel instead of the normalized inner product kernel, the derived kernel at each layer can be written explicitly as a linear combination of the derived kernel at the previous layer, that is,

$$K_{m+1}(f,g) = \sum_{j=1}^{|H_m|} \sum_{k=1}^{|H_m|} \omega_j \, \omega_k \, K_m(f \circ h_j, \; g \circ h_k) \quad \text{for all } f, g \in \mathrm{Im}(v_{m+1}) \ .$$

Proceeding recursively to apply this relation at every layer, in the end we find that the derived kernel K_{m+1} is a linear combination of the derived kernel K_1 at the first layer, for each $1 \leq m \leq n-1$. For example, in the case when Ψ is the average pooling function, we can write

$$K_{m+1}(f,g) = \frac{1}{|\widetilde{H}_m|^2} \sum_{h,h' \in \widetilde{H}_m} K_1(f \circ h, \; g \circ h') \quad \text{for all } f, g \in \mathrm{Im}(v_{m+1}) \ ,$$

where $\widetilde{H}_m$ is the set of all composed transformations to layer $m+1$,

$$\widetilde{H}_m = \{h_m \circ h_{m-1} \circ \cdots \circ h_1 \mid h_i \in H_i \text{ for } 1 \leq i \leq m\} \ ,$$

and $|\widetilde{H}_m|$ is the cardinality of $\widetilde{H}_m$,

$$|\widetilde{H}_m| = \prod_{i=1}^{m} |H_i| \ .$$

Hence in this case the linear architecture is equivalent to a single-layer network.

5 Analysis in a One-Dimensional Case

One of the most intriguing questions about hierarchical architectures is whether and to what extent the hierarchy gives rise to discrimination and invariance properties. Yet, even though the tradeoff between discrimination and invariance is at the heart of many hierarchical algorithms in practice, so far there is little theoretical understanding about the precise effect of hierarchical computations to the discrimination and invariance properties of the output. A first step in quantifying this effect was taken in [56], in which it was shown that in the case of one-dimensional strings, an exhaustive derived kernel architecture discriminates input strings up to reversal and checkerboard patterns.

In this section we generalize this analysis on strings and study the discrimination and invariance properties of the derived kernel in a more general architecture. We will demonstrate several ways to build an architecture that induces reversal invariance on K_n. We will also characterize the equivalence classes of the derived kernel in the case of a discriminative architecture. We hope that our analysis in this section can provide an insight into the emergence of discrimination and invariance in general hierarchical architectures, in particular in the case of two-dimensional images.

5.1 Setting and Preliminaries

We first describe the framework that we are working with in this section. Given a nonempty finite alphabet S, a k-string is a function $f\colon \{1,\dots,k\} \to S$, and we write $f = a_1 \dots a_k$ to indicate $f(i) = a_i \in S$ for $1 \le i \le k$. For example, in the case of binary strings we have $S = \{0,1\}$, and in the case of DNA sequences we have $S = \{A, C, G, T\}$.

We are working with an n-layer architecture, $n \in \mathbb{N}$, with the max pooling function and normalized inner product kernel. The patches are of the form $v_m = \{1,\dots,|v_m|\}$ for some $|v_m| \in \mathbb{N}$ with $|v_m| < |v_{m+1}|$. We assume that the function spaces consist of all possible strings of the appropriate size,

$$\mathrm{Im}(v_m) = \{f \mid f\colon v_m \to S\} = S^{|v_m|} \ .$$

Corresponding to the use of the normalized inner product kernel, the templates of the first kind at each layer are of the form $\widehat{N}_m(t)$ for some $t \in \mathrm{Im}(v_m)$, where $\widehat{N}_m(t)$ denotes the normalized neural response of t,

$$\widehat{N}_m(t) = \frac{N_m(t)}{\|N_m(t)\|} \ .$$

But in general, in this section we will also use templates of the second kind.

The discussion on the transformation sets H_m requires some notations. For $1 \le m \le n-1$, let L_m denote the set of all possible translations $h\colon v_m \to v_{m+1}$. That is,

$$L_m = \{h_1, \dots, h_{|v_{m+1}|-|v_m|+1}\} \ ,$$

where for $1 \le i \le |v_{m+1}| - |v_m| + 1$, the translation $h_i\colon v_m \to v_{m+1}$ is given by

$$h_i(j) = j + i - 1 \quad \text{for } 1 \le j \le |v_m| \ .$$

Thus, each translation $h_i \in L_m$ shifts the domain $v_m = \{1,\dots,|v_m|\}$ to $\{i, i+1, \dots, i+|v_m|-1\} \subseteq v_{m+1}$. The translations in L_m will be the primary example of transformations that we are going to use in this section, although at times we will also consider a more general set of transformations. Next, consider the following concept of the counterpart of a translation.

Definition 5.1 (Counterpart)**.** *Let $1 \le m \le n-1$. Given a translation $h_i \in L_m$, the counterpart of h_i is*

$$\chi(h_i) = h_{|v_{m+1}|-|v_m|+2-i} \ .$$

Similarly, given a set of translations $L \subseteq L_m$, the counterpart of L is

$$\chi(L) = \{\chi(h) \mid h \in L\} \ .$$

Note that $\chi(h) \in L_m$ if $h \in L_m$ and $\chi(L) \subseteq L_m$ if $L \subseteq L_m$. Also note that χ is an involution, that is, $\chi(\chi(h)) = h$ for $h \in L_m$ and $\chi(\chi(L)) = L$ for $L \subseteq L_m$.

Finally, the initial feature map is denoted by $N_1\colon \mathrm{Im}(v_1) \to \ell^2 \setminus \{0\}$. Since every such N_1 induces an initial kernel $K_1\colon \mathrm{Im}(v_1) \times \mathrm{Im}(v_1) \to [-1,1]$ given by

$$K_1(f,g) = \frac{\langle N_1(f), N_1(g)\rangle}{\|N_1(f)\|\,\|N_1(g)\|} \quad \text{for } f,g \in \mathrm{Im}(v_1) \ ,$$

in the following we will consider the properties of K_1 instead of N_1. The primary example of K_1 that we are going to use is

$$K_1(f,g) = 1 - \frac{d_H(f,g)}{|v_1|} \quad \text{for } f,g \in \mathrm{Im}(v_1) \ , \tag{8}$$

where $d_H(f,g)$ is the Hamming distance between f and g that counts the number of times f and g differ. Equivalently, we can write K_1 as

$$K_1(f,g) = \frac{\#\big\{1 \le i \le |v_1| \ \mid f(i) = g(i)\big\}}{|v_1|} \ .$$

This choice of K_1 corresponds to the initial feature map $N_1 \colon \mathrm{Im}(v_1) \to \mathbb{R}^{|v_1| \times |S|}$ given by

$$N_1(f)(i,a) = \begin{cases} 1 & \text{if } f(i) = a \\ 0 & \text{else} \end{cases} ,$$

for all $f \in \mathrm{Im}(v_1)$, $1 \le i \le |v_1|$, and $a \in S$.

Example 5.2 (Exhaustive architecture)**.** The analysis of Smale et al. in [56] considers an *exhaustive architecture*, which is an architecture where:

1. the patch sizes are $|v_m| = m$, for $1 \le m \le n$,
2. the template sets $\mathcal{T}_m$ consist of all possible templates of the first kind,
3. the transformation sets $H_m = L_m$ consist of all possible translations, and
4. the initial kernel K_1 is given by (8).

In this case we have the following explicit characterization of the equivalence classes of the derived kernel.

Theorem 5.3 (Theorem 4.1 in [56])**.** *In an exhaustive architecture, $K_n(f,g) = 1$ if and only if either $f = g$, f is the reversal of g, or f and g have the checkerboard pattern: $f = ababab\ldots$ and $g = bababa\ldots$ for some $a, b \in S$.*

In the next two sections we investigate the generalization of this result to a more general setting. In particular, we will see that the reversal and checkerboard patterns in the equivalence classes of K_n in Theorem 5.3 arise from essentially different origins: the reversal invariance of K_n is induced by the reversal invariance of K_1, while the checkerboard pattern emerges because of the hierarchical structure of the architecture.

In our analysis, we shall use the following preliminary result for an architecture with an exhaustive set of templates of the first kind. This result, which is a refinement of Proposition 4.1 in [56], states that the derived kernel and the neural response measure the same equivalences.

Proposition 5.4. *Consider an architecture with $n \ge 2$ layers with an exhaustive set of templates of the first kind at each layer,*

$$\mathcal{T}_m = \{\widehat{N}_m(t) \mid t \in Im(v_m)\} \quad \text{for all } 1 \le m \le n-1 \ .$$

Given $2 \le m \le n$ and $f, g \in Im(v_m)$, we have $K_m(f,g) = 1$ if and only if $N_m(f) = N_m(g)$.

We remark that, as noted in Remark 2.5, when we use exhaustive templates of the first kind we treat the template set as a multiset. That is, we keep all the encoded strings $\widehat{N}_m(t)$ in $\mathcal{T}_m$ even when some of them are duplicated, so that $\mathcal{T}_m$ contains $|S|^{|v_m|}$ templates.

5.2 Reversal Invariance of the Derived Kernel

In this section we study the reversal invariance of the derived kernel K_n. We are particularly interested in how the parameters of the architecture influence the reversal invariance of K_n. Before launching ourselves into the discussion, we first formalize the notions of reversal and reversal invariance.

For $k \in \mathbb{N}$, let $r_k \colon \{1, \ldots, k\} \to \{1, \ldots, k\}$ denote the reversal operation,

$$r_k(i) = k + 1 - i \quad \text{for } 1 \le i \le k \ .$$

Given a string $f = a_1 \ldots a_k \in S^k$, the *reversal* of f is $f \circ r \equiv f \circ r_k = a_k \ldots a_1 \in S^k$. In the development that follows, we will often write r to denote r_k if the domain size k is clear from context. We will explicitly write r_k only when it is necessary to distinguish the domain $\{1, \ldots, k\}$. Next, a kernel is reversal invariant if it cannot distinguish a string from its reversal.

Definition 5.5 (Reversal invariance)**.** *For $1 \leq m \leq n$, we say that the derived kernel K_m is reversal invariant if*

$$K_m(f, f \circ r) = 1 \quad \text{for all } f \in Im(v_m) \ .$$

We now look at several ways to build a hierarchical architecture where the derived kernel K_n is reversal invariant.

5.2.1 Reversal Invariance from the Initial Kernel

Our first approach is to start with an initial kernel K_1 that is reversal invariant and try to use this to induce reversal invariance on K_n. Indeed, as Smale et al. showed in [56], we can propagate invariance up in the hierarchy provided that the set of transformations H_m is rich enough. In our case, the condition on H_m turns out to be that each H_m consists of some translations and their counterparts. This condition arises because given a translation $h \in H_m$ and a string $f \in \mathrm{Im}(v_{m+1})$, the substring of f obtained from composing with $\chi(h)$ is the reversal of the substring of $f \circ r$ obtained from composing with h. This relation then allows us to propagate reversal invariance from one layer to the next.

Theorem 5.6. *Consider an architecture with $n \in \mathbb{N}$ layers. Suppose that at each layer $1 \leq m \leq n-1$ the transformation set H_m is a set of translations that is closed under taking counterparts, that is,*

$$H_m \subseteq L_m \quad \text{and} \quad \chi(H_m) = H_m \ .$$

If K_1 is reversal invariant, then K_n is reversal invariant.

In particular, if we take H_m to be the set of all transformations L_m then we automatically satisfy the hypothesis of Theorem 5.6. Therefore we can start with $|v_1| = 1$ and use the initial kernel K_1 given by (8), which is reversal invariant when $|v_1| = 1$, to obtain a reversal invariant K_n. Note that Theorem 5.6 does not make any assumption on the template sets $\mathcal{T}_m$, and in particular we can also use templates of the second kind.

5.2.2 Reversal Invariance from the Transformations

Another approach that we can pursue to build a reversal invariant K_n is to encode the information regarding the reversal operation in the transformation sets H_m. Specifically, instead of considering only translations in H_m, we can also consider reversals of translations. In effect, when comparing two strings, the architecture will compare not only their substrings, but also the reversals of their substrings. It is then easy to show that the resulting derived kernel will always be reversal invariant, without any additional assumptions on the initial kernel or the template sets. Moreover, it turns out that it suffices to make the assumption on the transformation set only at layer $n-1$, as the following result shows.

Theorem 5.7. *Consider an architecture with $n \geq 2$ layers. Suppose that the transformation set H_{n-1} consists of some translations and their reversals, that is, H_{n-1} is of the form*

$$H_{n-1} = L \cup \big(\chi(L) \circ r\big) \quad \text{for some } L \subseteq L_{n-1} \ ,$$

where $\chi(L) \circ r$ is the set of the reversals of the translations in $\chi(L)$,

$$\chi(L) \circ r \equiv \{h \circ r \mid h \in \chi(L)\} \ .$$

Then K_n is reversal invariant.

5.2.3 Reversal Invariance from the Templates

The third approach to obtain reversal invariance on K_n is to encode the notion of reversals in the set of templates $\mathcal{T}_m$. One reasonable attempt at constructing such templates is to require each $\mathcal{T}_m$ to be *symmetric* with respect to the reversal operation. More precisely, given a template of the second kind

$$\tau = \sum_{i=1}^{d} c_i \widehat{N}_m(t_i) \ ,$$

where $d \in \mathbb{N}$, $c_1, \ldots, c_d \in \mathbb{R}$, and $t_1, \ldots, t_d \in \mathrm{Im}(v_m)$, let $\tau \circ r$ denote the template that is the reversal of τ,

$$\tau \circ r = \sum_{i=1}^{d} c_i \widehat{N}_m(t_i \circ r) \ .$$

Then we can try to assume that we only use templates that are themselves invariant to reversals, that is,

$$\tau \circ r = \tau \quad \text{for all } \tau \in \mathcal{T}_m \ .$$

By doing so, we hope that the architecture can learn from these templates and induce reversal invariance on K_n. Indeed this is the case, as we shall show in Theorem 5.9. However, before discussing this result in more detail, we need to introduce the following concept of reversal symmetry.

Definition 5.8 (Reversal symmetry)**.** *For $1 \leq m \leq n$, we say that the derived kernel K_m is reversal symmetric if*

$$K_m(f, g) = K_m(f \circ r,\ g \circ r) \quad \textit{for all } f, g \in \mathit{Im}(v_m) \ .$$

Contrast this with the definition of reversal invariance, which can be equivalently formulated as the following condition,

$$K_m(f, g) = K_m(f \circ r,\ g) \quad \text{for all } f, g \in \mathrm{Im}(v_m) \ .$$

That is, K_m is reversal symmetric if we can reverse both input strings simultaneously without changing the output. On the other hand, K_m is reversal invariant if we can reverse each input string independently and still preserve the kernel value. Clearly reversal invariance implies reversal symmetry, but in general the converse is not true. For example, the initial kernel K_1 given by (8) is reversal symmetric but not reversal invariant, unless $|v_1| = 1$.

Theorem 5.9. *Consider an architecture with $n \geq 2$ layers. Suppose that at each layer $1 \leq m \leq n-1$ the transformation set H_m is a set of translation that is closed under taking counterparts,*

$$H_m \subseteq L_m \quad \textit{and} \quad \chi(H_m) = H_m \ ,$$

and the template set $\mathcal{T}_m$ consists of symmetric templates,

$$\tau \circ r = \tau \quad \textit{for all } \tau \in \mathcal{T}_m \ .$$

If K_1 is reversal symmetric, then K_n is reversal invariant.

We note that Theorem 5.9 is presented in full generality with templates of the second kind. If we prefer to stay within the domain of templates of the first kind, we can restrict ourselves to use the normalized neural responses of *palindrome* strings, which are strings $f \in \mathrm{Im}(v_m)$ with the property that $f \circ r = f$. Considering the initial kernel K_1 given by (8), which is reversal symmetric, in this case we have the following corollary.

Corollary 5.10. *Consider an architecture with $n \geq 2$ layers, where $H_m = L_m$ and the template sets are*

$$\mathcal{T}_m \subseteq \left\{ \widehat{N}_m(f) \mid f \in \mathit{Im}(v_m),\ f \circ r = f \right\} \ .$$

If the initial kernel K_1 is given by (8), then K_n is reversal invariant.

5.2.4 Impossibility of Learning Reversal Invariance with Exhaustive Templates

In Sections 5.2.1, 5.2.2, and 5.2.3, we have seen how to learn reversal invariance on K_n by appropriately choosing the parameters of the architecture. Given these encouraging results, we can start asking the converse question: if we know that K_n is reversal invariant, what can we say about the architecture? Equivalently, we can try to investigate to what extent the hypotheses of the results are necessary. For example, in the hypothesis of Theorem 5.9, instead of requiring the templates to be symmetric, can we still derive the same conclusion if we require the template sets to be symmetric? That is, can we learn reversal invariance only by assuming that $\tau \circ r \in \mathcal{T}_m$ whenever $\tau \in \mathcal{T}_m$?

The answer, as it turns out, is no; essentially, this is because the symmetry of the template sets does not sufficiently capture the notion of reversal invariance. In fact, we will show that when we take $\mathcal{T}_m$ to be the set of all possible templates of the first kind, which satisfies the symmetry property that we just proposed, the derived kernel K_n cannot be reversal invariant unless K_1 is also reversal invariant. One way to interpret this result is to say that the templates contribute to the discrimination power of the architecture. By using a more exhaustive set of templates, the architecture will be discriminative to more patterns. In particular, if the template set contains some patterns and their reversals, then the architecture will be discriminative to the reversal operation.

In the following result, the choice of $\mathcal{T}_m$ to be the exhaustive set of templates of the first kind allows us to propagate reversal non-invariance up in the hierarchy.

Theorem 5.11. *Consider an architecture with $n \in \mathbb{N}$ layers. Suppose that at each layer $1 \leq m \leq n-1$ the transformation set H_m is taken to be the exhaustive set of translations L_m, and the template set $\mathcal{T}_m$ is taken to be the exhaustive set of templates of the first kind,*

$$\mathcal{T}_m = \{\widehat{N}_m(t) \mid t \in Im(v_m)\} \ .$$

Suppose further that either one of the following conditions is true:

(1) $|v_{m+1}| - |v_m| = 1$ for $1 \leq m \leq n-1$, or

(2) K_1 is reversal symmetric.

If K_n is reversal invariant, then K_1 is reversal invariant.

As a concrete example of the result above, note that the kernel K_1 given by (8) is reversal invariant if and only if $|v_1| = 1$. Then Theorem 5.6 together with Theorem 5.11 give us the following corollary.

Corollary 5.12. *Consider an architecture with $n \in \mathbb{N}$ layers, where $H_m = L_m$ is the set of all possible translations and $\mathcal{T}_m$ is the set of all possible templates of the first kind. With the choice of the initial kernel K_1 given by (8), the derived kernel K_n is reversal invariant if and only if $|v_1| = 1$.*

5.3 Equivalence Classes of the Derived Kernel

We now turn our attention to the discrimination power of the derived kernel. That is, given $f, g \in \mathrm{Im}(v_n)$ such that $K_n(f,g) = 1$, what can we say about f and g? We are particularly interested in the analysis of the equivalence classes of the derived kernel that arise because of the intrinsic recursive structure of the hierarchical architecture. For this purpose, in this section we consider an architecture where:

1. the transformation set H_m is the set of all possible translations L_m, for $1 \leq m \leq n-1$;
2. the template set $\mathcal{T}_m$ is the set of all possible templates of the first kind, for $1 \leq m \leq n-1$; and
3. the initial kernel K_1 is *fully discriminative*, that is,

$$K_1(f,g) = 1 \quad \text{if and only if} \quad f = g \ .$$

For example, the kernel K_1 given by (8) is fully discriminative. Having made these choices, the only remaining free parameter is the patch sizes $|v_m|$.

We note that when $|v_1| = 1$, a fully discriminative K_1 is still reversal invariant since in this case $f \circ r = f$ for all $f \in \mathrm{Im}(v_1)$. Theorem 5.6 then tells us that K_n is also reversal invariant. So we now consider the case when $|v_1| > 1$, so that K_1 is no longer reversal invariant. It turns out, as we shall see in Theorem 5.13, that in this case K_n can discriminate input strings up to periodic patterns, provided that $|v_1|$ is sufficiently large.

To state the theorem more precisely, we first need the following definitions. Given $2 \leq k < p$, we say that a p-string $f = a_1 \dots a_p$ is k-periodic if f is periodic with period k, that is, $a_i = a_{i+k}$ for $1 \leq i \leq p-k$. Note, however, that saying f is k-periodic does not imply that k is the smallest period of f. In particular, if f is k-periodic, then f is $k\ell$-periodic for any $\ell \in \mathbb{N}$ for which $\ell k < p$. Next, given two p-strings f and g, we write $f \overset{(k)}{\sim} g$ if f and g are k-periodic and g is some shifted version of f, that is, there exists $1 \leq j \leq k-1$ such that $g(i) = f(i+j)$ for all $1 \leq i \leq k$. Note that $f \overset{(2)}{\sim} g$ means f and g have the checkerboard pattern.

Theorem 5.13. *Consider an architecture with $n \geq 2$ layers. Let $\ell \in \mathbb{N}$ be the maximum jump in the patch sizes of the subsequent layers,*

$$\ell = \max_{2 \leq m \leq n} \left(|v_m| - |v_{m-1}| \right) \ .$$

If $|v_1| \geq 3\ell + 2$, then $K_n(f,g) = 1$ if and only if $f = g$ or $f \overset{(k)}{\sim} g$ for some $2 \leq k \leq \ell + 1$.

We note that the bound on $|v_1|$ in the theorem above is not tight. Wibisono [61] showed that the conclusion of Theorem 5.13 holds for $|v_1| \geq 2$ when $\ell = 1$, and for $|v_1| \geq 3$ when $\ell = 2$. However, the proof in [61] is considerably more complicated and involves an exhaustive case analysis. By only considering the case when $|v_1|$ is large enough, we are able to provide a more elegant proof for the general case.

Intuitively, the condition on the lower bound of $|v_1|$ is necessary for the conclusion to hold because when the initial patch sizes are too small, the periodic patterns interfere with each other to yield new equivalence classes which are hard to characterize. When $|v_1|$ is sufficiently large, however, the periodic patterns separate into distinct equivalence classes that we can identify explicitly. The shifted periodic patterns in the equivalence classes of K_n can be interpreted as the translation invariance that one hopes to obtain from the pooling operations in the hierarchy, and the periodicity of the patterns emerges because we are working with finite function patches.

5.4 Mirror Symmetry in Two-Dimensional Images

In the proof of Theorem 5.11 (case (2)), we have also proved the following result regarding the propagation of reversal symmetry.

Proposition 5.14. *Consider an architecture with $n \geq 2$ layers. Suppose at layer $1 \leq m \leq n-1$ we know that K_m is reversal symmetric. Then given $\tau \in \mathcal{T}_m$ with $\tau \circ r \in \mathcal{T}_m$, we have*

$$N_{m+1}(f)(\tau) = N_{m+1}(f \circ r_{|v_{m+1}|})(\tau \circ r_{|v_m|}) \quad \text{for all } f \in Im(v_{m+1}) \ .$$

In particular, if the template set $\mathcal{T}_m$ is closed under the reversal operation, that is, if $\tau \in \mathcal{T}_m$ implies $\tau \circ r \in \mathcal{T}_m$, then K_{m+1} is also reversal symmetric.

In this section we extend the notion of reversal symmetry to the case of two-dimensional arrays, and discuss the analog of the result above in the case of mirror symmetry. The framework that we are considering is the two-dimensional version of the string setting previously described. That is, we now work with square lattices as patches, and consider functions from these patches to a finite alphabet. For example, we can describe an image as a function from the pixel locations to a finite set of pixel values, say integers between 0 and 255, representing the grayscale values.

Formally, let S be a nonempty finite alphabet. We are working with an n-layer architecture, $n \in \mathbb{N}$, with the max pooling function and normalized inner product kernel. The patches are two-dimensional square arrays $v_m = \{1, \dots, |v_m|\} \times \{1, \dots, |v_m|\}$, where $|v_m| < |v_{m+1}|$. We assume that the function spaces consist of all possible images of the appropriate size,

$$\mathrm{Im}(v_m) = \{f \mid f \colon v_m \to S\} \ .$$

We also assume that the transformation set H_m consists of all possible translations $h \colon v_m \to v_{m+1}$, that is,

$$H_m = \{h_{i,j} \mid 1 \le i, j \le |v_{m+1}| - |v_m| + 1\}$$

where for each $1 \le i, j \le |v_{m+1}| - |v_m| + 1$, the translation $h_{i,j} \colon v_m \to v_{m+1}$ is given by

$$h_{i,j}(k, \ell) = (k + i - 1, \ \ell + j - 1) \quad \text{for } 1 \le k, \ell \le |v_m| \ .$$

We can see that $f \circ h$ is a patch of the image f, for $f \in \mathrm{Im}(v_{m+1})$ and $h \in H_m$.

Thinking of the first and second coordinates of each patch as the x and y-axis, respectively, we now define the analog of the reversal operation on images, which is the reflection about the y-axis. Given $k \in \mathbb{N}$, let $r_k \colon \{1, \dots, k\} \times \{1, \dots, k\} \to \{1, \dots, k\} \times \{1, \dots, k\}$ denote the reflection operation,

$$r_k(i, j) = (k + 1 - i, j) \quad \text{for } 1 \le i, j \le k \ .$$

Given an image $f \colon \{1, \dots, k\} \times \{1, \dots, k\} \to S$, the *reflection* of f is $f \circ r \equiv f \circ r_k$. Intuitively, the reflection of an image is what we see when we put the image in front of a mirror. We then introduce the following concept of reflection symmetry, which we also call *mirror symmetry*.

Definition 5.15 (Reflection symmetry)**.** *For $1 \le m \le n$, we say that the derived kernel K_m is reflection symmetric if*

$$K_m(f, g) = K_m(f \circ r, g \circ r) \quad \textit{for all } f, g \in \mathit{Im}(v_m) \ .$$

We can show that the result of Proposition 5.14 extends naturally to the case of reflection symmetry.

Proposition 5.16. *Consider an architecture with $n \ge 2$ layers. Suppose at layer $1 \le m \le n - 1$ we know that K_m is reflection symmetric. Then given $\tau \in \mathcal{T}_m$ with $\tau \circ r \in \mathcal{T}_m$, we have*

$$N_{m+1}(f)(\tau) = N_{m+1}(f \circ r_{|v_{m+1}|})(\tau \circ r_{|v_m|}) \quad \textit{for all } f \in \mathit{Im}(v_{m+1}) \ .$$

In particular, if the template set $\mathcal{T}_m$ is closed under the reflection operation, that is, if $\tau \in \mathcal{T}_m$ implies $\tau \circ r \in \mathcal{T}_m$, then K_{m+1} is also reflection symmetric.

A corollary to the result above that is of special interest is the following, which says that a reflection-symmetric template induces the same response in an image and its reflection.

Corollary 5.17. *In the setting of Proposition 5.16, if $\tau \in \mathcal{T}_m$ has the property that $\tau = \tau \circ r$, then*

$$N_{m+1}(f)(\tau) = N_{m+1}(f \circ r_{|v_{m+1}|})(\tau) \quad \textit{for all } f \in \mathit{Im}(v_{m+1}) \ .$$

5.5 Discussion

The analysis on strings provides a starting point toward understanding the invariance of a hierarchical architecture. Through the results in this section, we have seen a glimpse of how the discrimination and invariance properties of the architecture interact with the different choices of the parameters. These results suggest two main avenues that we can pursue to continue our work.

Further analysis on one-dimensional strings. The case of one-dimensional strings is a relatively fruitful frontier because the analysis is simple and tractable. In spite of our initial attempt at exploring this landscape, there are still many questions left to be resolved. For example:

- In Section 5.2 we have seen three main methods to induce reversal invariance on K_n: via the initial kernel K_1, the transformation sets H_m, and the template sets $\mathcal{T}_m$. What other ways are there to build a reversal invariant K_n? What can we say about the architecture when K_n is reversal invariant? A preliminary attempt at answering the latter question is provided in Theorem 5.11.

- Another interesting question is related to the robustness of the derived kernel to small transformations, that is, the *approximate invariance* of K_n. For example, if we have two strings $f, g \in \mathrm{Im}(v_n)$ that are not equivalent under K_n but they only differ in one or few letters, what can we say about $K_n(f, g)$? This question is partially motivated by applications to biology and DNA analysis, where the bases of the DNA constantly undergo mutations. This analysis might also be of relevance in the case of binary strings, because in practical situations some of the bits are corrupted due to transmissions through noisy channels.

Extension to two-dimensional images. One way of working with images is to treat them as two-dimensional strings with a finite set of values, for example using the finite-precision representation of real values on a computer. This viewpoint should allow us to extend some of the results in this section to the case of images. In particular, as alluded to at the end of Section 5.3, the shifted periodic patterns in Theorem 5.13 might correspond to the grating patterns that are commonly observed in the topographic filter maps when the architecture is specifically built to incorporate translation invariance [26, 28]. Moreover, in the case of images the domain will be a two-dimensional grid $\{1, \dots, k\} \times \{1, \dots, k\}$, and thus we expect the invariance properties to involve more complex transformations, including the symmetry group of the square, the dihedral group D_4. The discussion in Section 5.4 is an initial step in this direction.

However, this treatment of images as two-dimensional strings is not without limitation because it ignores the algebraic structure of the set of values, namely the real (or rational) numbers. In particular, real numbers have an ordering relation and arithmetic operations. These structures will affect the choice of the initial kernel, as well as what constitutes an invariance. For example, we might want to consider invariance under multiplying the values of an image by a constant, which can be interpreted as looking at the same image under different lighting conditions.

6 Conclusions and Open Questions

Conclusions. In this paper we proposed a theoretical framework of hierarchical architectures that generalizes the definition introduced by Smale et al. [56]. This generalized definition of the neural response and derived kernel allows us to study a wider class of hierarchical algorithms, including simplified versions of the feedforward hierarchical model [50, 54] and convolutional neural networks [26, 31, 32]. We then used this framework to study the following theoretical properties of hierarchical architectures:

- *Range compression.* We showed that hierarchical architectures with nonnegative values and normalization procedures are susceptible to range compression; essentially, the derived kernel becomes increasingly saturated at each layer. This saturation effect presents a challenging problem in practice because it can lead to decreased performance of the architecture in some empirical tasks. We proposed several methods to mitigate this range compression effect and demonstrated that these methods succeeded in restoring the performance of the architecture.

- *Linear architectures.* In the case of a linear architecture, we proved that the dimensionality of the neural response at each layer is restricted by the dimensionality at the first layer. Moreover, we showed that when we use an orthonormal basis of the neural responses as templates, the derived

kernel is independent of the choice of basis. In particular, when we use the inner product kernel, the architecture collapses into a single-layer linear computation.

- *One-dimensional strings.* We investigated the discrimination and invariance properties of the derived kernel when the input data are one-dimensional strings. We showed how to induce reversal invariance in the hierarchy by appropriately choosing the model parameters. Furthermore, we proved that it is impossible to learn reversal invariance when we use an exhaustive set of templates. Finally, we characterized the equivalence classes of the derived kernel in the case of a discriminative architecture.

These results represent an exciting and promising beginning of a rigorous theory of hierarchical architectures. We hope that our results provide some insight into the theoretical properties of hierarchical algorithms, and will be useful for guiding the development of hierarchical algorithms in the future.

Open questions. Thus far our analysis has been focused on the *intrinsic* properties of hierarchical architectures, namely, properties that are inherent in the architectures and do not depend on external tasks. On the other hand, one could also investigate the *extrinsic* properties of hierarchical architectures. For example, given a classification problem, we could look into the performance of the architecture when we use the derived kernel as the similarity measure for classification. Studying the extrinsic properties of hierarchical architectures is more difficult than studying the intrinsic properties because there might not be a universally correct answer; instead, as Bengio argued in [3], the optimal answer might depend on the specific task at hand.

Nevertheless, the extrinsic properties of hierarchical architectures are more relevant to the problems in practice, and, as such, need to be investigated theoretically. Several questions in this direction that we believe are of immediate interest are:

- *Number of layers.* What is the optimal number of layers in the architecture? Can we show that having more layers is always better in terms of performance? Hinton et al. [22] showed that learning an extra layer in a deep belief network increases the lower bound of the log-likelihood of the data. Surprisingly, many hierarchical algorithms in practice are running with relatively few layers [26, 37, 54]. We believe that a theoretical analysis on the effect of the number of layers in the architecture can give some insight into this problem.

- *Parameter choices.* How should we choose the parameters of the model to achieve a good performance? For example, once we fix the number of layers in the architecture, how should we choose the patch sizes at each layer? In the case of image classification on the MNIST digit dataset, Smale et al. [56] suggested that there is an optimal configuration of patch sizes for a 3-layer architecture. From our results in this paper, Figure 4 also suggests that there is an optimal configuration of patch sizes for a 4-layer architecture. Moreover, the performance results in [56] and Figure 4 seem to exhibit similar up-and-down patterns as the patch sizes vary. This leads to the question of whether we can characterize the optimal patch sizes theoretically.

- *Sample complexity.* How does the hierarchical architecture influence the sample complexity of the problem? As previously mentioned, the human brain is exceptionally skilled at learning and generalizing from only a small subset of examples. Poggio and Smale [46] argued that this capability might be due to the hierarchical structure of the cortex, and that hierarchical architectures might also have the advantage of low sample complexity. Therefore, it will be interesting to study this question from a theoretical perspective using the framework of the neural response and derived kernel.

A Appendix: A Theoretical Framework of Hierarchical Architectures

Proof of Proposition 2.7. From the the weak boundedness of Ψ and the admissibility of $\mathcal{T}_m$, we have

$$\begin{aligned}
\|N_{m+1}(f)\|^2 &= \sum_{\tau\in\mathcal{T}_m} \big|\Psi\big(\langle\Phi(N_m(f\circ h)),\,\tau\rangle\big)\big|^2 \\
&\le C_{|H_m|}^2 \sum_{\tau\in\mathcal{T}_m}\sum_{h\in H_m} \big\langle\Phi\big(N_m(f\circ h)\big),\,\tau\big\rangle^2 \\
&= C_{|H_m|}^2 \sum_{h\in H_m}\sum_{\tau\in\mathcal{T}_m} \big\langle\Phi\big(N_m(f\circ h)\big),\,\tau\big\rangle^2 \\
&\le C_{|H_m|}^2\, C^2 \sum_{h\in H_m} \big\|\Phi\big(N_m(f\circ h)\big)\big\|^2 \\
&< \infty\ .
\end{aligned}$$

Note that the last inequality holds because we assume H_m to be finite. □

B Appendix: Range Compression

In this section we present the proofs of the results from Section 3.

B.1 Setting and Preliminaries

B.1.1 Strongly Bounded Pooling Functions

Proof of Lemma 3.2. Let $\zeta\colon \mathbb{R}_0 \to \mathbb{R}_0$ be a non-decreasing concave function that dominates Ψ. We claim that $\Psi\circ\sigma$ is dominated by $\zeta\circ\sigma$. Clearly $\zeta\circ\sigma$ is nonnegative since both ζ and σ are nonnegative. Moreover, since ζ and σ are non-decreasing and concave, for all $x, y \in \mathbb{R}_0$ and $0 \le \lambda \le 1$ we have

$$\big(\zeta\circ\sigma\big)\big(\lambda x + (1-\lambda)y\big) \ge \zeta\big(\lambda\,\sigma(x) + (1-\lambda)\,\sigma(y)\big) \ge \lambda\,\big(\zeta\circ\sigma\big)(x) + (1-\lambda)\,\big(\zeta\circ\sigma\big)(y)\ ,$$

which means that $\zeta\circ\sigma$ is a concave function.

Now, using the fact that σ is non-decreasing, for each $n\in\mathbb{N}$ and $\alpha = (\alpha_1,\ldots,\alpha_n)\in\mathbb{R}^n$ we have

$$\big(\Psi\circ\sigma\big)(\alpha) \le C_n\,\zeta\Big(\max_{1\le i\le n}\sigma(\alpha_i)\Big) = C_n\,\big(\zeta\circ\sigma\big)\Big(\max_{1\le i\le n}\alpha_i\Big)\ .$$

Similarly,

$$\big(\Psi\circ\sigma\big)(\alpha) \ge C_n\,\zeta\Big(\min_{1\le i\le n}\sigma(\alpha_i)\Big) = C_n\,\big(\zeta\circ\sigma\big)\Big(\min_{1\le i\le n}\alpha_i\Big)\ .$$

This shows that $\Psi\circ\sigma$ is a strongly bounded pooling function that is dominated by $\zeta\circ\sigma$, as desired. □

B.1.2 Dynamic Range of the Derived Kernel

We first state the following preliminary results that will be helpful to prove Theorem 3.4 and Theorem 3.6.

Lemma B.1. *Let $\xi = (\xi_1,\ldots,\xi_k)$ and $\omega = (\omega_1,\ldots,\omega_k)$ be two vectors in the positive orthant of $\mathbb{R}^k$, that is, assume $\xi_i,\ \omega_i > 0$ for $1\le i\le k$. Let θ denote the angle between ξ and ω. If we hold ξ fixed and vary ω along $\omega_1\in(0,\infty)$, then $\cos\theta$ has one extremum point, which is a maximum. In particular, if ω_1 lies in an interval $[r_1, r_2]$, then $\cos\theta$ is minimized at either $\omega_1 = r_1$ or $\omega_1 = r_2$.*

Proof. Recall that

$$\cos\theta = \frac{\langle \xi, \omega \rangle}{\|\xi\| \, \|\omega\|} = \frac{\sum_{i=1}^{k} \xi_i \omega_i}{\left(\sum_{i=1}^{k} \xi_i^2\right)^{1/2} \left(\sum_{i=1}^{k} \omega_i^2\right)^{1/2}} \ .$$

Since we hold everything fixed except for ω_1, we can consider $\cos\theta$ as a function of ω_1 only. For simplicity, write

$$\cos\theta = \frac{A\omega_1 + B}{C(\omega_1^2 + D)^{1/2}} \ ,$$

where

$$A = \xi_1, \quad B = \sum_{i=2}^{k} \xi_i \omega_i, \quad C = \left(\sum_{i=1}^{k} \xi_i^2\right)^{1/2}, \quad \text{and} \quad D = \sum_{i=2}^{k} \omega_i^2 \ .$$

Note that $A, B, C, D > 0$. Take the derivative of $\cos\theta$ with respect to ω_1,

$$\begin{aligned} \frac{d\cos\theta}{d\omega_1} &= \frac{A(\omega_1^2 + D)^{1/2} - (A\omega_1 + B)\omega_1(\omega_1^2 + D)^{-1/2}}{C(\omega_1^2 + D)} \\ &= \frac{A(\omega_1^2 + D) - (A\omega_1 + B)\omega_1}{C(\omega_1^2 + D)^{3/2}} \\ &= \frac{AD - B\omega_1}{C(\omega_1^2 + D)^{3/2}} \ . \end{aligned}$$

Setting the derivative to 0, we see that $\cos\theta$ has an extremum point at $\omega_1 = AD/B$. Since $d\cos\theta/d\omega_1 > 0$ for $\omega_1 < AD/B$ and $d\cos\theta/d\omega_1 < 0$ for $\omega_1 > AD/B$, we conclude that $\omega_1 = AD/B$ is a maximum point of $\cos\theta$, as desired.

□

Using the preceding lemma, we can prove the following result.

Lemma B.2. *Let $b \geq a > 0$, and let $\mathcal{C}$ be the hypercube*

$$\mathcal{C} = \left\{ x = (x_1, \ldots, x_k) \in \mathbb{R}^k \mid a \leq x_i \leq b \ \text{ for } \ i = 1, \ldots, k \right\} \ .$$

Then

$$\inf_{x,y \in \mathcal{C}} \langle\, \widehat{x}, \widehat{y} \,\rangle \geq \frac{2ab}{a^2 + b^2} \ .$$

Proof. Let $\mathcal{F} \colon \mathcal{C} \times \mathcal{C} \to \mathbb{R}$ be the function $\mathcal{F}(x, y) = \langle\, \widehat{x}, \widehat{y} \,\rangle$. Since $\mathcal{F}$ is continuous and $\mathcal{C}$ is a compact subset of $\mathbb{R}^k$, by the extreme value theorem we know that $\mathcal{F}$ achieves its minimum in $\mathcal{C}$, say at (x', y'). Let θ denote the angle between x' and y', and note that we can write $\mathcal{F}(x', y') = \cos\theta$. Lemma B.1 then tells us that x' and y' must be some vertices of $\mathcal{C}$. This is because if one of the points, say x', is not a vertex, then it has a coordinate x_i such that $a < x_i < b$. By holding y' fixed and setting x_i to either a or b we can decrease the value of $\cos\theta$, thus contradicting the minimality of (x', y').

Now that we know (x', y') is a vertex of $\mathcal{C}$, we can write $x' = (x_1, \ldots, x_k)$ and $y' = (y_1, \ldots, y_k)$ with $x_1, \ldots, x_k, y_1, \ldots, y_k \in \{a, b\}$. Define

$$\begin{aligned} \alpha &= \#\{1 \leq i \leq k \mid x_i = b, \ y_i = a\}, \\ \beta &= \#\{1 \leq i \leq k \mid x_i = a, \ y_i = b\}, \\ \gamma &= \#\{1 \leq i \leq k \mid x_i = b, \ y_i = b\}, \quad \text{and} \\ \delta &= \#\{1 \leq i \leq k \mid x_i = a, \ y_i = a\} \ . \end{aligned}$$

Clearly $\alpha, \beta, \gamma, \delta \geq 0$ and $\alpha + \beta + \gamma + \delta = k$. This allows us to write

$$\begin{aligned}\left\langle \widehat{x'}, \widehat{y'} \right\rangle &= \frac{\sum_{i=1}^k x_i y_i}{\left(\sum_{i=1}^k x_i^2\right)^{1/2} \left(\sum_{i=1}^k y_i^2\right)^{1/2}} \\ &= \frac{\delta a^2 + (\alpha+\beta)ab + \gamma b^2}{\left((\beta+\delta)a^2 + (\alpha+\gamma)b^2\right)^{1/2} \left((\alpha+\delta)a^2 + (\beta+\gamma)b^2\right)^{1/2}} \ .\end{aligned}$$

By the arithmetic-geometric mean inequality,

$$\left((\beta+\delta)a^2 + (\alpha+\gamma)b^2\right)^{1/2} \left((\alpha+\delta)a^2 + (\beta+\gamma)b^2\right)^{1/2} \ \leq \ \frac{(\alpha+\beta+2\delta)a^2 + (\alpha+\beta+2\gamma)b^2}{2} \ ,$$

and hence

$$\left\langle \widehat{x'}, \widehat{y'} \right\rangle \geq \frac{2\left(\delta a^2 + (\alpha+\beta)ab + \gamma b^2\right)}{(\alpha+\beta+2\delta)a^2 + (\alpha+\beta+2\gamma)b^2} \ .$$

Intuitively, we expect the angle of separation θ to be maximized when x' and y' are two opposite vertices of $\mathcal{C}$. This corresponds to the choice $\gamma = \delta = 0$, in which case the right hand side of the inequality above becomes $2ab/(a^2+b^2)$. Indeed, we can check that

$$\begin{aligned}&\frac{2\left(\delta a^2 + (\alpha+\beta)ab + \gamma b^2\right)}{(\alpha+\beta+2\delta)a^2 + (\alpha+\beta+2\gamma)b^2} - \frac{2ab}{a^2+b^2} \\ &\quad= \frac{2\Big(\left(\delta a^2 + (\alpha+\beta)ab + \gamma b^2\right)\left(a^2+b^2\right) - ab\left((\alpha+\beta+2\delta)a^2 + (\alpha+\beta+2\gamma)b^2\right)\Big)}{\left((\alpha+\beta+2\delta)a^2 + (\alpha+\beta+2\gamma)b^2\right)\left(a^2+b^2\right)} \\ &\quad= \frac{2\Big(\left(\delta a^4 + (\alpha+\beta)a^3b + (\gamma+\delta)a^2b^2 + (\alpha+\beta)ab^3 + \gamma b^4\right) - \left((\alpha+\beta+2\delta)a^3b + (\alpha+\beta+2\gamma)ab^3\right)\Big)}{\left((\alpha+\beta+2\delta)a^2 + (\alpha+\beta+2\gamma)b^2\right)\left(a^2+b^2\right)} \\ &\quad= \frac{2\left(\delta a^4 - 2\delta a^3 b + (\gamma+\delta)a^2b^2 - 2\gamma ab^3 + \gamma b^4\right)}{\left((\alpha+\beta+2\delta)a^2 + (\alpha+\beta+2\gamma)b^2\right)\left(a^2+b^2\right)} \\ &\quad= \frac{2\left(\delta a^2(b-a)^2 + \gamma b^2(b-a)^2\right)}{\left((\alpha+\beta+2\delta)a^2 + (\alpha+\beta+2\gamma)b^2\right)\left(a^2+b^2\right)} \\ &\quad= \frac{2(\delta a^2 + \gamma b^2)(b-a)^2}{\left((\alpha+\beta+2\delta)a^2 + (\alpha+\beta+2\gamma)b^2\right)\left(a^2+b^2\right)} \\ &\quad\geq 0 \ .\end{aligned}$$

This shows that $\left\langle \widehat{x'}, \widehat{y'} \right\rangle \geq 2ab/(a^2+b^2)$, as desired. □

Finally, the following lemma connects the dynamic range of the derived kernel at each layer with the normalized inner product between the neural responses at the next layer.

Lemma B.3. *Consider a nonnegative architecture with a strongly bounded pooling function and convex templates. At each layer $m \in \mathbb{N}$ we have*

$$\widehat{K}_{m+1}(f,g) \geq \frac{2\delta(K_m)}{1+\delta(K_m)^2} \quad \text{for all } f, g \in Im(v_{m+1}) \ .$$

Proof. Let

$$a = \inf_{f,g \in \mathrm{Im}(v_m)} K_m(f,g) \quad \text{and} \quad b = \sup_{f,g \in \mathrm{Im}(v_m)} K_m(f,g) \ ,$$

so that $\delta(K_m) = a/b$. Note that $a \geq 0$ since we are working with a nonnegative architecture. If $a = 0$ then $2\delta(K_m)/(1+\delta(K_m)^2) = 0$ and there is nothing to prove, so assume $a > 0$. Let $\zeta\colon \mathbb{R}_0 \to \mathbb{R}_0$ be a non-decreasing concave function that dominates the pooling function Ψ.

Let $f \in \mathrm{Im}(v_{m+1})$. Given a template $\tau \in \mathcal{T}_m$, write

$$\tau = \sum_{i=1}^{d} c_i \Phi(N_m(f_i)) \quad \text{where } d \in \mathbb{N},\ f_i \in \mathrm{Im}(v_m),\ \text{ and } c_i \geq 0 \text{ with } \sum_{i=1}^{d} c_i = 1 \ .$$

Then for every transformation $h \in H_m$ we have

$$a = \sum_{i=1}^{d} c_i a \ \leq \ \sum_{i=1}^{d} c_i K_m(f \circ h,\ f_i) = \langle \Phi(N_m(f \circ h)),\ \tau \rangle \leq \sum_{i=1}^{d} c_i b = b \ ,$$

and since Ψ is strongly bounded,

$$C_{|H_m|}\, \zeta(a) \leq N_{m+1}(f)(\tau) = \Psi\big(\langle \Phi(N_m(f \circ h)),\ \tau \rangle\big) \leq C_{|H_m|}\, \zeta(b) \ .$$

This means the neural response $N_{m+1}(f)$ lies in the $|\mathcal{T}_m|$-dimensional hypercube

$$\mathcal{C} = \Big\{ x \in \mathbb{R}^{|\mathcal{T}_m|} \ \Big|\ C_{|H_m|}\, \zeta(a) \leq x_i \leq C_{|H_m|}\, \zeta(b) \ \text{ for } i = 1, \ldots, |\mathcal{T}_m| \Big\} \ .$$

Lemma B.2 then tells us that for every $f, g \in \mathrm{Im}(v_{m+1})$, we have the lower bound

$$\widehat{K}_{m+1}(f,g) \geq \inf_{x,y \in \mathcal{C}} \langle\, \widehat{x},\ \widehat{y}\, \rangle \geq \frac{2\, C^2_{|H_m|}\, \zeta(a)\, \zeta(b)}{C^2_{|H_m|}\, \zeta(a)^2 + C^2_{|H_m|}\, \zeta(b)^2} = \frac{2\big(\zeta(a)/\zeta(b)\big)}{1 + \big(\zeta(a)/\zeta(b)\big)^2} \ .$$

Now note that since ζ is a concave function,

$$\zeta(a) = \zeta\left(\left(1 - \frac{a}{b}\right) \cdot 0 + \frac{a}{b} \cdot b\right) \geq \left(1 - \frac{a}{b}\right) \zeta(0) + \frac{a}{b}\, \zeta(b) \geq \frac{a}{b}\, \zeta(b) = \delta(K_m)\, \zeta(b) \ ,$$

which gives us $\zeta(a)/\zeta(b) \geq \delta(K_m)$. Therefore, since $x \mapsto 2x/(1+x^2)$ is an increasing function on $0 \leq x \leq 1$, we conclude that

$$\widehat{K}_{m+1}(f,g) \geq \frac{2\delta(K_m)}{1 + \delta(K_m)^2} \ ,$$

as desired. □

Given Lemma B.3, Theorem 3.4 then follows easily.

Proof of Theorem 3.4. If $\delta(K_m) \to 1$, then Lemma B.3 implies that we also have

$$\lim_{m \to \infty} \inf_{f,g \in \mathrm{Im}(v_m)} \widehat{K}_m(f,g) = 1 \ .$$

Therefore, noting that for all $f, g \in \mathrm{Im}(v_m)$ we can write

$$\|\widehat{N}_m(f) - \widehat{N}_m(g)\|^2 = 2 - 2\widehat{K}_m(f,g) \ ,$$

we then conclude that

$$\lim_{m \to \infty} \sup_{f,g \in \mathrm{Im}(v_m)} \|\widehat{N}_m(f) - \widehat{N}_m(g)\| \ = \ \sqrt{2 - 2 \lim_{m \to \infty} \inf_{f,g \in \mathrm{Im}(v_m)} \widehat{K}_m(f,g)} \ = \ 0 \ .$$

□

B.2 Range Compression: Normalized Kernel Functions

Proof of Theorem 3.6. We begin by squaring the Lipschitz continuity condition in (3) and using the assumption that K is normalized to obtain

$$K(x,y) \geq \langle\, \widehat{x}, \widehat{y}\,\rangle \quad \text{for all } x, y \in \ell^2 \setminus \{0\} \ .$$

Given $f, g \in \mathrm{Im}(v_{m+1})$, we can substitute $x = N_{m+1}(f)$ and $y = N_{m+1}(g)$ to the inequality above and use Lemma B.3 to get

$$K_{m+1}(f,g) \geq \widehat{K}_{m+1}(f,g) \geq \frac{2\delta(K_m)}{1+\delta(K_m)^2} \ .$$

Since this holds for all $f, g \in \mathrm{Im}(v_{m+1})$, we then conclude that

$$\delta(K_{m+1}) = \inf_{f,g \in \mathrm{Im}(v_{m+1})} K_{m+1}(f,g) \geq \frac{2\delta(K_m)}{1+\delta(K_m)^2} \ .$$

Now to show the convergence assertion, we note that the inequality above implies

$$1 \geq \delta(K_n) \geq \Lambda^{(n-m)}\big(\delta(K_m)\big) \quad \text{for } n \geq m+1 \ , \tag{9}$$

where $\Lambda\colon [0,1] \to [0,1]$ is the function $\Lambda(a) = 2a/(1+a^2)$, and $\Lambda^{(n-m)}$ is the composition of Λ with itself $n-m$ times. Observe that Λ is an increasing function and $a \leq \Lambda(a) \leq 1$ for $0 \leq a \leq 1$, with equality occurring when $a = 0$ or $a = 1$ (see Figure 6). This means the mapping $a \mapsto \Lambda(a)$ only has two fixed points, $a = 0$ and $a = 1$, and the point $a = 0$ is unstable. Once we have $a > 0$, the value of a under this mapping will increase monotonically until it converges to $a = 1$. Therefore, from (9) we see that if $\delta(K_m) > 0$ then $\Lambda^{(n-m)}\big(\delta(K_m)\big) \to 1$ as $n \to \infty$, and therefore,

$$\lim_{n\to\infty} \delta(K_n) = 1 \ ,$$

as desired.

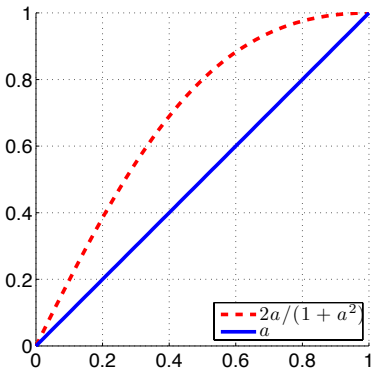


Figure 6: Plot of the function $\Lambda(a) = \frac{2a}{1+a^2}$.

□

B.3 Range Compression: Normalized Pooling Functions

We first show that given Definition 3.9, a sigmoid function approaches its asymptote at a certain rate. Specifically, we claim that the derivative $\sigma'(x)$ approaches 0 faster than $1/x$. Since $\sigma(x)$ has a finite limit as $x \to \infty$, this will then imply that

$$F_\sigma(x) = \frac{2x\sigma'(x)}{\sigma(x)} \to 0 \quad \text{as } x \to \infty \ ,$$

as claimed in Section 3.3.

Lemma B.4. *Given a sigmoid function σ,*

$$\lim_{x\to\infty} x\sigma'(x) = 0 \ .$$

Proof. Since σ is non-decreasing and concave on $\mathbb{R}_0$, σ' is nonnegative and non-increasing on $\mathbb{R}_0$. Furthermore, we know that σ' is integrable on $\mathbb{R}_0$,

$$\int_0^\infty \sigma'(x)\,dx \ = \ \lim_{x\to\infty} \sigma(x) - \sigma(0) \ < \ \infty \ .$$

Now suppose the contrary that

$$\limsup_{x\to\infty} x\sigma'(x) = c > 0 \ .$$

Then there exists a sequence $\{x_n\}_{n\in\mathbb{N}}$ in $\mathbb{R}_0$ such that $x_n \to \infty$ and $x_n\sigma'(x_n) > c/2$ for all $n \in \mathbb{N}$. Let $\{y_n\}_{n\in\mathbb{N}}$ be a subsequence of $\{x_n\}_{n\in\mathbb{N}}$ such that $y_{n+1} \geq (n+1)y_n/n$ for all $n \in \mathbb{N}$. Since σ' is non-increasing,

$$\sigma'(x) \geq \sigma'(y_{n+1}) > \frac{c}{2y_{n+1}} \quad \text{if } y_n < x \leq y_{n+1} \ .$$

This implies

$$\int_{y_1}^\infty \sigma'(x)\,dx \ = \ \sum_{n=1}^\infty \int_{y_n}^{y_{n+1}} \sigma'(x)\,dx \ > \ \sum_{n=1}^\infty \frac{c}{2}\frac{(y_{n+1}-y_n)}{y_{n+1}} \ \geq \ \frac{c}{2}\sum_{n=1}^\infty \left(1 - \frac{n}{n+1}\right) \ = \ \frac{c}{2}\sum_{n=1}^\infty \frac{1}{n+1} \ = \ \infty \ ,$$

contradicting the integrability of σ'. Thus we must have

$$\limsup_{x\to\infty} x\sigma'(x) \leq 0 \ .$$

Finally, since $\sigma'(x) \geq 0$ for $x \in \mathbb{R}_0$, we conclude that

$$\lim_{x\to\infty} x\sigma'(x) = 0 \ ,$$

as desired. □

We add two remarks.

Remark B.5. The proof of Lemma B.4 above can be extended to show that $x\log(x)\sigma'(x) \to 0$ as $x \to \infty$, so $x\sigma'(x)$ must eventually decrease faster than $1/\log(x)$. This gives the upper bound $\varepsilon_{\eta,\sigma} = \mathcal{O}(e^{1/\eta})$ as $\eta \to 0$, which is achieved, for example, when σ is given by

$$\sigma(x) = \frac{1}{\log k} - \frac{1}{\log(x+k)} \quad \text{for some } k > 1 \ .$$

However, as we saw in Section 3.3, for common sigmoid functions the values of $\varepsilon_{\eta,\sigma}$ are small, which makes Theorem 3.10 applicable in practice.

Remark B.6. We note that the assumption that σ is non-decreasing and concave on $\mathbb{R}_0$ is necessary to conclude Lemma B.4. To see this, suppose we do not assume σ to be non-decreasing or concave, so that σ' does not have to be nonnegative and non-increasing. Take σ defined by $\sigma(0) = 0$ and[g]

$$\sigma'(x) = (-1)^{\lfloor x \rfloor} \frac{1}{x+1} \quad \text{for } x \geq 0 \ ,$$

where $\lfloor x \rfloor$ is the largest integer less than or equal to x. Clearly σ' is still integrable on $\mathbb{R}_0$, for example by decomposing the integral into a sum of integrals on the intervals $[n, n+1)$ and then using the alternating series test. However, in this case

$$x\sigma'(x) \ = \ (x+1)\sigma'(x) - \sigma'(x) \ = \ (-1)^{\lfloor x \rfloor} - \sigma'(x)$$

does not converge, since $(-1)^{\lfloor x \rfloor}$ is alternating between 1 and -1, and $\sigma'(x) \to 0$ as $x \to \infty$. Next, suppose we assume that σ is non-decreasing but we do not assume that σ is concave. This means σ' is nonnegative but it is allowed to oscillate. Consider σ defined by $\sigma(0) = 0$ and[h]

$$\sigma'(x) = \begin{cases} \frac{1}{x+1} & \text{if } x \in \mathbb{N}, \\ \frac{1}{(x+1)^2} & \text{if } x \in \mathbb{R}_0 \setminus \mathbb{N} \ . \end{cases}$$

Clearly σ' is still integrable on $\mathbb{R}_0$, so σ has a horizontal asymptote. However, we see that when $x \in \mathbb{N}$,

$$x\sigma'(x) = \frac{x}{x+1} \to 1 \quad \text{as } x \to \infty \ ,$$

so clearly $x\sigma'(x)$ does not converge to 0.

We now prove Theorem 3.10.

Proof of Theorem 3.10. Let

$$a = \inf_{f,g \in \mathrm{Im}(v_m)} K_m(f,g) \quad \text{and} \quad b = \sup_{f,g \in \mathrm{Im}(v_m)} K_m(f,g) \ ,$$

so that $\delta(K_m) = a/b$. Note that we assume $a \geq \varepsilon_{\eta,\sigma} \geq 0$. If $a = 0$ then there is nothing to prove, so assume $a > 0$.

Let $f \in \mathrm{Im}(v_{m+1})$. Given a template $\tau \in \mathcal{T}_m$, recall that we are using the inner product kernel, and write

$$\tau = \sum_{i=1}^{d} c_i N_m(f_i) \quad \text{where } \ d \in \mathbb{N}, \ f_i \in \mathrm{Im}(v_m), \ \text{ and } \ c_i \geq 0 \ \text{ with } \ \sum_{i=1}^{d} c_i = 1 \ .$$

Then for every transformation $h \in H_m$ we have

$$a = \sum_{i=1}^{d} c_i a \ \leq \ \sum_{i=1}^{d} c_i K_m(f \circ h, \ f_i) = \langle N_m(f \circ h), \ \tau \rangle \leq \sum_{i=1}^{d} c_i b = b \ ,$$

and since σ is a non-decreasing function,

$$\sigma(a) \leq \sigma\big(\langle N_m(f \circ h), \ \tau \rangle\big) \leq \sigma(b) \ .$$

Therefore, the strong boundedness of Ψ gives us

$$C_{|H_m|}\, \zeta\big(\sigma(a)\big) \leq N_{m+1}(f)(\tau) = \big(\Psi \circ \sigma\big)\big(\langle N_m(f \circ h), \ \tau \rangle\big) \leq C_{|H_m|}\, \zeta\big(\sigma(b)\big) \ .$$

[g] Or take σ' to be a continuous approximation of the proposed function above.
[h] Or, as before, we can take σ' to be a continuous approximation of the proposed function.

This gives us a bound on the value of the derived kernel,

$$|\mathcal{T}_m|\, C_{|H_m|}^2\, \zeta\big(\sigma(a)\big)^2 \le K_{m+1}(f,g) \le |\mathcal{T}_m|\, C_{|H_m|}^2\, \zeta\big(\sigma(b)\big)^2 \quad \text{for all } f,g \in \mathrm{Im}(v_{m+1}) \ . \tag{10}$$

Therefore, we can also bound the dynamic range

$$\delta(K_{m+1}) \ge \frac{|\mathcal{T}_m|\, C_{|H_m|}^2\, \zeta\big(\sigma(a)\big)^2}{|\mathcal{T}_m|\, C_{|H_m|}^2\, \zeta\big(\sigma(b)\big)^2} = \frac{\zeta\big(\sigma(a)\big)^2}{\zeta\big(\sigma(b)\big)^2} \ .$$

As we saw in the proof of Lemma B.3, since ζ is concave we have

$$\frac{\zeta\big(\sigma(a)\big)}{\zeta\big(\sigma(b)\big)} \ge \frac{\sigma(a)}{\sigma(b)} \ ,$$

and thus

$$\delta(K_{m+1}) \ge \frac{\sigma(a)^2}{\sigma(b)^2} \ .$$

Now to conclude that

$$\delta(K_{m+1}) \ge \delta(K_m)^\eta = \frac{a^\eta}{b^\eta} \ ,$$

it suffices to show that the function

$$S(x) := \frac{\sigma(x)^2}{x^\eta}$$

is non-increasing for $x \ge a$. Computing the derivative of S, we obtain

$$S'(x) = \frac{2\sigma(x)\sigma'(x)x^\eta - \eta\sigma(x)^2 x^{\eta-1}}{x^{2\eta}} = \frac{\sigma(x)\big[2x\sigma'(x) - \eta\sigma(x)\big]}{x^{\eta+1}} \ .$$

Therefore, from the definition of $\varepsilon_{\eta,\sigma}$ we see that $S'(x) \le 0$ for $x \ge \varepsilon_{\eta,\sigma}$. Since $a \ge \varepsilon_{\eta,\sigma}$ by assumption, this means S is non-increasing for $x \ge a$ and hence $S(a) \ge S(b)$, as desired.

Finally, if condition (7) holds, then from (10) we see that

$$K_{m+1}(f,g) \ge \frac{\varepsilon_{\eta,\sigma}}{C_{|H_m|}^2\, \zeta\big(\sigma(\varepsilon_{\eta,\sigma})\big)^2}\, C_{|H_m|}^2\, \zeta\big(\sigma(a)\big)^2 \ge \varepsilon_{\eta,\sigma} \quad \text{for all } f,g \in \mathrm{Im}(v_{m+1}) \ .$$

In particular, if (7) holds at all layers $n \ge m$, then since $0 < \eta < 1$,

$$\delta(K_n) \ge \delta(K_m)^{\eta^{n-m}} \to 1 \quad \text{as } n \to \infty \ .$$

This completes the proof of Theorem 3.10. □

C Appendix: Linear Architectures

In this section we present the proofs from Section 4.

C.1 Rank Constraint

Proof of Theorem 4.1. Let $1 \le m \le n-1$. If $\dim(\mathcal{N}_m) = \infty$ then there is nothing to prove, so assume that $\dim(\mathcal{N}_m) = p < \infty$. Let $(\phi_1, \ldots, \phi_p)$ be an orthonormal basis of $\mathcal{N}_m$. Note that each normalized

neural response $\widehat{N}_m(f)$ lies in $\mathcal{N}_m$, so we can write $\widehat{N}_m(f)$ as a linear combination of $\phi_1, \ldots, \phi_p$ for all $f \in \mathrm{Im}(v_m)$. Furthermore, using the linearity of Ψ we can find $\omega = (\omega_1, \ldots, \omega_{|H_m|}) \in \mathbb{R}^{|H_m|}$ such that

$$\Psi(\alpha) = \langle \alpha,\, \omega \rangle = \sum_{i=1}^{|H_m|} \alpha_i \omega_i \quad \text{for all } \alpha = (\alpha_1, \ldots, \alpha_{|H_m|}) \in \mathbb{R}^{|H_m|} \ .$$

Now, given $f \in \mathrm{Im}(v_{m+1})$ and $\tau \in \mathcal{T}_m$ we can write

$$N_{m+1}(f)(\tau) = \Psi\big(\langle \widehat{N}_m(f \circ h),\, \tau \rangle\big) = \sum_{i=1}^{|H_m|} \omega_i \, \langle \widehat{N}_m(f \circ h_i),\, \tau \rangle \ .$$

Since $\widehat{N}_m(f \circ h_i) \in \mathcal{N}_m$ and $\tau \in \mathcal{T}_m \subseteq \mathcal{N}_m$, we can expand the inner product in terms of the orthonormal basis of $\mathcal{N}_m$ as follows,

$$\langle \widehat{N}_m(f \circ h_i),\, \tau \rangle = \sum_{j=1}^{p} \langle \widehat{N}_m(f \circ h_i),\, \phi_j \rangle \, \langle \tau,\, \phi_j \rangle \ .$$

Substituting this summation to the original expression and rearranging the terms give us

$$N_{m+1}(f)(\tau) = \sum_{i=1}^{|H_m|} \omega_i \left(\sum_{j=1}^{p} \langle \widehat{N}_m(f \circ h_i),\, \phi_j \rangle \, \langle \tau,\, \phi_j \rangle \right) = \sum_{j=1}^{p} \left(\sum_{i=1}^{|H_m|} \omega_i \, \langle \widehat{N}_m(f \circ h_i),\, \phi_j \rangle \right) \langle \tau,\, \phi_j \rangle \ .$$

Since this holds for all $\tau \in \mathcal{T}_m$, we can write

$$N_{m+1}(f) = \sum_{j=1}^{p} \left(\sum_{i=1}^{|H_m|} \omega_i \, \langle \widehat{N}_m(f \circ h_i),\, \phi_j \rangle \right) \ \varphi_j \ ,$$

where for $j = 1, \ldots, p$ we define $\varphi_j \in \ell^2(\mathcal{T}_m)$ by

$$\varphi_j(\tau) = \langle \tau,\, \phi_j \rangle \quad \text{for all } \tau \in \mathcal{T}_m \ .$$

Note that φ_j is well defined because by the admissibility condition of $\mathcal{T}_m$ we have

$$\|\varphi_j\|^2 = \sum_{\tau \in \mathcal{T}_m} \langle \tau,\, \phi_j \rangle^2 \leq C \, \|\phi_j\|^2 = C < \infty \quad \text{for } i = 1, \ldots, p \ .$$

The computation above shows that for every $f \in \mathrm{Im}(v_{m+1})$ we can write $N_{m+1}(f)$ as a linear combination of $\{\varphi_1, \ldots, \varphi_p\}$. This means $\{\varphi_1, \ldots, \varphi_p\}$ spans $\mathcal{N}_{m+1}$, and hence we conclude that

$$\dim(\mathcal{N}_{m+1}) \leq p = \dim(\mathcal{N}_m) \ ,$$

as desired. □

C.2 Basis Independence

Proof of Theorem 4.2. Given $1 \leq m \leq n-1$, we will show that the inner product between $N_{m+1}(f)$ and $N_{m+1}(g)$ is independent of the choice of the orthonormal basis of $\mathcal{N}_m$ in $\mathcal{T}_m$, for all $f, g \in \mathrm{Im}(v_{m+1})$. This will then imply the desired conclusion that the derived kernel

$$K_{m+1}(f, g) = \frac{\langle N_{m+1}(f),\, N_{m+1}(g) \rangle}{\sqrt{\langle N_{m+1}(f),\, N_{m+1}(f) \rangle \, \langle N_{m+1}(g),\, N_{m+1}(g) \rangle}}$$

is also independent of the choice of the orthonormal basis of $\mathcal{N}_m$.

Let $(\phi_1, \ldots, \phi_p)$ be the orthonormal basis of $\mathcal{N}_m$ that $\mathcal{T}_m$ takes as templates of the second kind, where $p \leq \infty$. Note that for each $f \in \mathrm{Im}(v_m)$ we have $\widehat{N}_m(f) \in \mathcal{N}_m$, so $(\phi_1, \ldots, \phi_p)$ is also an orthonormal basis of the normalized neural responses.

Next, since the pooling function is linear, we can find $\omega = (\omega_1, \ldots, \omega_{|H_m|}) \in \mathbb{R}^{|H_m|}$ such that

$$\Psi(\alpha) = \langle \alpha,\, \omega \rangle = \sum_{i=1}^{|H_m|} \alpha_i \omega_i \quad \text{for all } \alpha = (\alpha_1, \ldots, \alpha_{|H_m|}) \in \mathbb{R}^{|H_m|} \ .$$

Then for every $f \in \mathrm{Im}(v_{m+1})$ and $1 \leq i \leq p$, we have

$$N_{m+1}(f)(\phi_i) = \Psi\big(\langle \widehat{N}_m(f \circ h),\, \phi_i \rangle\big) = \sum_{j=1}^{|H_m|} \omega_j \, \langle \widehat{N}_m(f \circ h_j),\, \phi_i \rangle \ .$$

Now, given $f, g \in \mathrm{Im}(v_{m+1})$ we can compute

$$\begin{aligned}
\big\langle N_{m+1}(f),\, N_{m+1}(g) \big\rangle &= \sum_{i=1}^{p} N_{m+1}(f)(\phi_i)\, N_{m+1}(g)(\phi_i) \\
&= \sum_{i=1}^{p} \left(\sum_{j=1}^{|H_m|} \omega_j \, \langle \widehat{N}_m(f \circ h_j),\, \phi_i \rangle \right) \left(\sum_{k=1}^{|H_m|} \omega_k \, \langle \widehat{N}_m(g \circ h_k),\, \phi_i \rangle \right) \\
&= \sum_{j=1}^{|H_m|} \sum_{k=1}^{|H_m|} \omega_j \, \omega_k \left(\sum_{i=1}^{p} \langle \widehat{N}_m(f \circ h_j),\, \phi_i \rangle \, \langle \widehat{N}_m(g \circ h_k),\, \phi_i \rangle \right) \\
&= \sum_{j=1}^{|H_m|} \sum_{k=1}^{|H_m|} \omega_j \, \omega_k \, \langle \widehat{N}_m(f \circ h_j),\, \widehat{N}_m(g \circ h_k) \rangle \\
&= \sum_{j=1}^{|H_m|} \sum_{k=1}^{|H_m|} \omega_j \, \omega_k \, K_m(f \circ h_j,\, g \circ h_k) \ .
\end{aligned}$$

This shows that the inner product between $N_{m+1}(f)$ and $N_{m+1}(g)$ is independent of the choice of $(\phi_1, \ldots, \phi_p)$, as desired. □

D Appendix: Analysis in a One-Dimensional Case

In this section we present the proofs from Section 5.

D.1 Setting and Preliminaries

Proof of Proposition 5.4. Clearly if $N_m(f) = N_m(g)$ then we have

$$K_m(f, g) = \frac{\langle N_m(f), N_m(g) \rangle}{\|N_m(f)\| \, \|N_m(g)\|} = 1 \ .$$

Conversely, suppose that $K_m(f, g) = 1$. Since K is the normalized inner product kernel, this means the neural responses $N_m(f)$ and $N_m(g)$ are collinear, that is, $N_m(f) = cN_m(g)$ for some $c \geq 0$. Let $h \in H_{m-1}$, and consider the template $\tau = \widehat{N}_{m-1}(f \circ h) \in \mathcal{T}_{m-1}$. Note that on the one hand we have

$$N_m(f)(\tau) = \max_{h' \in H_{m-1}} \langle \widehat{N}_{m-1}(f \circ h'),\, \tau \rangle = \max_{h' \in H_{m-1}} K_{m-1}(f \circ h',\, f \circ h) = K_{m-1}(f \circ h,\, f \circ h) = 1 \ ,$$

while on the other hand,

$$N_m(g)(\tau) = \max_{h' \in H_{m-1}} \left\langle \widehat{N}_{m-1}(g \circ h'),\, \tau \right\rangle = \max_{h' \in H_{m-1}} K_{m-1}(g \circ h',\, f \circ h) \leq 1 \ .$$

Therefore, from the relation $N_m(f) = cN_m(g)$ we obtain

$$c \geq cN_m(g)(\tau) = N_m(f)(\tau) = 1 \ .$$

Similarly, by considering another template $\tau' = \widehat{N}_{m-1}(g \circ h) \in \mathcal{T}_{m-1}$ we see that $c \leq 1$. Hence we conclude that $c = 1$, and thus $N_m(f) = N_m(g)$. □

D.2 Reversal Invariance of the Derived Kernel

D.2.1 Reversal Invariance from the Initial Kernel

We note that the result of Theorem 5.6 actually follows immediately from Proposition 3.1 and Corollary 4.1 in [56]. However, for the sake of completeness we reconstruct the proof here because part of the argument will be needed to prove Theorem 5.7 and Theorem 5.9.

Proof of Theorem 5.6. It suffices to show that if K_m is reversal invariant then K_{m+1} is reversal invariant, for $1 \leq m \leq n-1$. First, from the assumption that K_m is reversal invariant we have

$$\widehat{N}_m(g) = \widehat{N}_m(g \circ r) \quad \text{for all } g \in \mathrm{Im}(v_m) \ .$$

Next, since H_m is closed under taking counterparts and χ is an involution, $\chi \colon H_m \to H_m$ is a bijection. Furthermore, given $h = h_i \in H_m$, we note that for $1 \leq j \leq |v_m|$ we have

$$\begin{aligned} \big(r_{|v_{m+1}|} \circ h_i\big)(j) &= r_{|v_{m+1}|}(i+j-1) \\ &= |v_{m+1}| + 1 - (i+j-1) \\ &= |v_{m+1}| + 2 - i - j \ , \end{aligned}$$

and similarly,

$$\begin{aligned} \big(\chi(h_i) \circ r_{|v_m|}\big)(j) &= \big(h_{|v_{m+1}|-|v_m|+2-i} \circ r_{|v_m|}\big)(j) \\ &= h_{|v_{m+1}|-|v_m|+2-i}(|v_m| + 1 - j) \\ &= |v_{m+1}| - |v_m| + 2 - i + (|v_m| + 1 - j) - 1 \\ &= |v_{m+1}| + 2 - i - j \ . \end{aligned}$$

This shows that

$$r_{|v_{m+1}|} \circ h = \chi(h) \circ r_{|v_m|} \quad \text{for all } h \in H_m \ .$$

Therefore, for each $f \in \mathrm{Im}(v_{m+1})$ and $\tau \in \mathcal{T}_m$ we have

$$\begin{aligned} N_{m+1}(f \circ r_{|v_{m+1}|})(\tau) &= \max_{h \in H_m} \left\langle \widehat{N}_m(f \circ r_{|v_{m+1}|} \circ h),\, \tau \right\rangle \\ &= \max_{h \in H_m} \left\langle \widehat{N}_m(f \circ \chi(h) \circ r_{|v_m|}),\, \tau \right\rangle \\ &= \max_{h \in H_m} \left\langle \widehat{N}_m(f \circ \chi(h)),\, \tau \right\rangle \\ &= N_{m+1}(f)(\tau) \ . \end{aligned}$$

This gives us $N_{m+1}(f \circ r) = N_{m+1}(f)$, and thus $K_{m+1}(f, f \circ r) = 1$. Since this holds for all $f \in \mathrm{Im}(v_{m+1})$, we conclude that K_{m+1} is reversal invariant. □

Remark D.1. By inspecting the proof above, we see that Theorem 5.6 also holds when we use a general pooling function Ψ that is invariant under permutations of its inputs. That is, we can replace the max pooling function in Theorem 5.6 with a pooling function Ψ satisfying the property that for each $k \in \mathbb{N}$,

$$\Psi(\alpha) = \Psi(\pi(\alpha)) \quad \text{for all } \alpha \in \mathbb{R}^k \text{ and } \pi \in \mathfrak{S}_k \ ,$$

where $\mathfrak{S}_k$ is the symmetric group of degree k, and given $\alpha = (\alpha_1, \ldots, \alpha_k) \in \mathbb{R}^k$ and $\pi \in \mathfrak{S}_k$,

$$\pi(\alpha) = (\alpha_{\pi(1)}, \ldots, \alpha_{\pi(k)}) \in \mathbb{R}^k \ .$$

For example, the average pooling function is invariant under permutations of its inputs.

D.2.2 Reversal Invariance from the Transformations

Proof of Theorem 5.7. Recall from the proof of Theorem 5.6 that for each $h \in L \subseteq L_{n-1}$ we have

$$r_{|v_n|} \circ h = \chi(h) \circ r_{|v_{n-1}|} \ .$$

Then, given $f \in \mathrm{Im}(v_n)$ we note that

$$\begin{aligned}
\left\{ f \circ r_{|v_n|} \circ h \mid h \in H_{n-1} \right\} &= \left\{ f \circ r_{|v_n|} \circ h \mid h \in L \right\} \cup \left\{ f \circ r_{|v_n|} \circ \chi(h) \circ r_{|v_{n-1}|} \mid h \in L \right\} \\
&= \left\{ f \circ \chi(h) \circ r_{|v_{n-1}|} \mid h \in L \right\} \cup \left\{ f \circ r_{|v_n|} \circ r_{|v_n|} \circ h \mid h \in L \right\} \\
&= \left\{ f \circ \chi(h) \circ r_{|v_{n-1}|} \mid h \in L \right\} \cup \left\{ f \circ h \mid h \in L \right\} \\
&= \left\{ f \circ h \mid h \in H_{n-1} \right\} \ .
\end{aligned}$$

Therefore, for each $\tau \in \mathcal{T}_{n-1}$ we have

$$N_n(f \circ r)(\tau) = \max_{h \in H_{n-1}} \left\langle \widehat{N}_{n-1}(f \circ r \circ h),\, \tau \right\rangle = \max_{h \in H_{n-1}} \left\langle \widehat{N}_{n-1}(f \circ h),\, \tau \right\rangle = N_n(f)(\tau) \ .$$

This gives us $N_n(f \circ r) = N_n(f)$, and thus $K_n(f,\, f \circ r) = 1$. Since this holds for all $f \in \mathrm{Im}(v_n)$, we conclude that K_n is reversal invariant. □

Remark D.2. As noted in Remark D.1, by inspecting the proof above we see that Theorem 5.7 also holds when we use a general pooling function Ψ that is invariant under permutations of its inputs.

D.2.3 Reversal Invariance from the Templates

Proof of Theorem 5.9. It suffices to show that if K_m is reversal symmetric then K_{m+1} is reversal invariant, for $1 \leq m \leq n-1$. The conclusion will then follow since we assume K_1 to be reversal symmetric, and as we noted above, reversal invariance implies reversal symmetry. First recall from the proof of Theorem 5.6 that $\chi \colon H_m \to H_m$ is a bijection and

$$r_{|v_{m+1}|} \circ h = \chi(h) \circ r_{|v_m|} \quad \text{for all } h \in H_m \ .$$

Let $f \in \mathrm{Im}(v_{m+1})$. Given a template $\tau \in \mathcal{T}_m$, write

$$\tau = \sum_{i=1}^{d} c_i \widehat{N}_m(t_i) \quad \text{for some } d \in \mathbb{N},\ c_i \in \mathbb{R},\ \text{and } t_i \in \mathrm{Im}(v_m) \ .$$

Then, using the reversal symmetry of K_m, we can write

$$N_{m+1}(f \circ r_{|v_{m+1}|})(\tau) = \max_{h \in H_m} \left\langle \widehat{N}_m(f \circ r_{|v_{m+1}|} \circ h),\, \tau \right\rangle$$

$$
\begin{aligned}
&= \max_{h \in H_m} \left\langle \widehat{N}_m(f \circ \chi(h) \circ r_{|v_m|}),\, \tau \right\rangle \\
&= \max_{h \in H_m} \left(\sum_{i=1}^{d} c_i \, K_m\big(f \circ \chi(h) \circ r_{|v_m|},\, t_i\big) \right) \\
&= \max_{h \in H_m} \left(\sum_{i=1}^{d} c_i \, K_m\big(f \circ \chi(h),\, t_i \circ r_{|v_m|}\big) \right) \\
&= \max_{h \in H_m} \left\langle \widehat{N}_m(f \circ \chi(h)),\, \tau \circ r_{|v_m|} \right\rangle \\
&= N_{m+1}(f)(\tau \circ r_{|v_m|}) \ .
\end{aligned}
$$

Since we assume τ is symmetric, we then have

$$
N_{m+1}(f \circ r)(\tau) = N_{m+1}(f)(\tau) \quad \text{ for all } \tau \in \mathcal{T}_m \ .
$$

This gives us $N_{m+1}(f) = N_{m+1}(f \circ r)$, and thus $K_{m+1}(f, f \circ r) = 1$. Since this holds for all $f \in \mathrm{Im}(v_{m+1})$, we conclude that K_{m+1} is reversal symmetric, as desired. □

Remark D.3. As noted in Remarks D.1 and D.2, by inspecting the proof above we see that Theorem 5.9 also holds when we use a general pooling function Ψ that is invariant under permutations of its inputs.

D.2.4 Impossibility of Learning Reversal Invariance with Exhaustive Templates

Proof of Theorem 5.11. In this proof we use of the following terminologies. Given a k-string $f = a_1 \dots a_k$, the *tail* of f is the longest posterior substring $a_i a_{i+1} \dots a_k$ with the property that $a_j = a_k$ for $i \le j \le k$. Let $\lambda(f)$ denote the length of the tail of f. Furthermore, given $1 \le m \le n-1$ and $f \in \mathrm{Im}(v_m)$, let $e_f \in \mathrm{Im}(v_{m+1})$ denote the *extension* of f obtained by extending the tail of f, that is,

$$
e_f(i) = \begin{cases} f(i) & \text{if } 1 \le i \le |v_m|, \text{ and} \\ f(|v_m|) & \text{if } |v_m| + 1 \le i \le |v_{m+1}| \ . \end{cases}
$$

For example, suppose $S = \{a, b, c, d\}$, $|v_1| = 5$, and $|v_2| = 8$. Given $f = abcdd \in \mathrm{Im}(v_1)$, the tail of f is dd and $\lambda(f) = 2$, and the extension of f is $e_f = abcddddd \in \mathrm{Im}(v_2)$.

We now consider two cases, corresponding to the two conditions in the statement of the theorem.

Case (1). Suppose $|v_{m+1}| - |v_m| = 1$ for $1 \le m \le n-1$. We will show that if K_m is not reversal invariant then K_{m+1} is not reversal invariant, for $1 \le m \le n-1$. This will imply that if K_1 is not reversal invariant then K_n is not reversal invariant, as desired.

Let $1 \le m \le n-1$, and assume K_m is not reversal invariant. Choose a *witness* for the failure of K_m to be reversal invariant, that is, a string $f \in \mathrm{Im}(v_m)$ such that $K_m(f, f \circ r) < 1$. We will use f to exhibit a witness for the failure of K_{m+1} to be reversal invariant. To this end, we claim that either:

(a) $K_{m+1}(e_f,\, e_f \circ r) < 1$, or

(b) we can find another string $f' \in \mathrm{Im}(v_m)$ with the property that $\lambda(f') > \lambda(f)$ and $K_m(f',\, f' \circ r) < 1$.

If we fall on Case (b), then we can repeat the procedure above with f' in place of f. However, observe that since $\lambda(f) \le |v_m|$ and every iteration of this procedure through Case (b) increases $\lambda(f)$, we must eventually land in Case (a), in which case we are done since e_f is a witness for the failure of K_{m+1} to be reversal invariant. Thus it suffices to prove the claim above.

Since $|v_{m+1}| - |v_m| = 1$, the set H_m only consists of two translations h_1 and h_2. Note that

$$\{e_f \circ h \mid h \in H_m\} = \{f,\ e_f \circ h_2\}$$

and

$$\{e_f \circ r_{|v_{m+1}|} \circ h \mid h \in H_m\} = \{e_f \circ h_2 \circ r_{|v_m|},\ f \circ r_{|v_m|}\} \ .$$

If $K_m(e_f \circ h_2 \circ r,\ f) < 1$, then with $\tau = \widehat{N}_m(f) \in \mathcal{T}_m$ we have

$$N_{m+1}(e_f \circ r_{|v_{m+1}|})(\tau) = \max\big\{K_m(e_f \circ h_2 \circ r_{|v_m|},\ f),\ K_m(f \circ r,\ f)\big\} < 1 \ ,$$

while

$$N_{m+1}(e_f)(\tau) = \max\big\{K_m(f, f),\ K_m(e_f \circ h_2,\ f)\big\} = K_m(f, f) = 1 \ .$$

This implies $N_{m+1}(e_f) \neq N_{m+1}(e_f \circ r)$, and hence $K_{m+1}(e_f,\ e_f \circ r) < 1$ by Proposition 5.4. Similarly, if $K_m(e_f \circ h_2,\ f \circ r) < 1$, then by considering $\tau' = \widehat{N}_m(f \circ r)$ we can show that $N_{m+1}(e_f)(\tau') < 1$ and $N_{m+1}(e_f \circ r)(\tau') = 1$, which implies $K_{m+1}(e_f,\ e_f \circ r) < 1$.

Now suppose that $K_m(e_f \circ h_2 \circ r,\ f) = 1$ and $K_m(e_f \circ h_2,\ f \circ r) = 1$. Then notice that we must have

$$K_m(e_f \circ h_2,\ e_f \circ h_2 \circ r) < 1 \ ,$$

for otherwise

$$N_m(f) = N_m\big(e_f \circ h_2 \circ r\big) = N_m\big(e_f \circ h_2\big) = N_m(f \circ r) \ ,$$

contradicting the fact that $K_m(f, f \circ r) < 1$. Since $\lambda(e_f \circ h_2) = \lambda(f) + 1 > \lambda(f)$, we can take the new witness f' to be $e_f \circ h_2 \in \mathrm{Im}(v_m)$, and we are done.

Case (2). Suppose K_1 is reversal symmetric. We will first show that if K_m is reversal symmetric then K_{m+1} is reversal symmetric, for $1 \leq m \leq n-1$. The assumption that K_1 is reversal symmetric will then imply that K_m is reversal symmetric for all $1 \leq m \leq n$.

Let $1 \leq m \leq n-1$, and suppose K_m is reversal symmetric. Recall from the proof of Theorem 5.6 that $\chi\colon H_m \to H_m$ is a bijection and

$$r_{|v_{m+1}|} \circ h = \chi(h) \circ r_{|v_m|} \qquad \text{for all } h \in H_m \ .$$

Then, given a string $f \in \mathrm{Im}(v_{m+1})$ and a template $\tau = \widehat{N}_m(g') \in \mathcal{T}_m$, where $g' \in \mathrm{Im}(v_m)$, we can write

$$\begin{aligned} N_{m+1}(f)(\tau) &= \max_{h \in H_m} \ K_m(f \circ h,\ g') \\ &= \max_{h \in H_m} \ K_m(f \circ h \circ r_{|v_m|},\ g' \circ r_{|v_m|}) \\ &= \max_{h \in H_m} \ K_m(f \circ r_{|v_{m+1}|} \circ \chi(h),\ g' \circ r_{|v_m|}) \\ &= N_{m+1}(f \circ r_{|v_{m+1}|})(\tau \circ r_{|v_m|}) \ . \end{aligned}$$

Note that the mapping

$$\mathcal{T}_m \ni \tau = \widehat{N}_m(g') \overset{r}{\longmapsto} \widehat{N}_m(g' \circ r) = \tau \circ r \in \mathcal{T}_m$$

is a bijection. This implies

$$\|N_{m+1}(f)\| = \|N_{m+1}(f \circ r)\| \ ,$$

and therefore,

$$\widehat{N}_{m+1}(f)(\tau) = \frac{N_{m+1}(f)(\tau)}{\|N_{m+1}(f)\|} = \frac{N_{m+1}(f \circ r_{|v_{m+1}|})(\tau \circ r_{|v_m|})}{\|N_{m+1}(f \circ r_{|v_{m+1}|})\|} = \widehat{N}_{m+1}(f \circ r_{|v_{m+1}|})(\tau \circ r_{|v_m|}) \ .$$

Hence, for all $f, g \in \mathrm{Im}(v_{m+1})$ we have

$$\begin{aligned} K_{m+1}(f,g) &= \langle \widehat{N}_{m+1}(f),\ \widehat{N}_{m+1}(g) \rangle \\ &= \sum_{\tau \in \mathcal{T}_m} \widehat{N}_{m+1}(f)(\tau)\ \widehat{N}_{m+1}(g)(\tau) \\ &= \sum_{\tau \in \mathcal{T}_m} \widehat{N}_{m+1}(f \circ r_{|v_{m+1}|})(\tau \circ r_{|v_m|})\ \widehat{N}_{m+1}(g \circ r_{|v_{m+1}|})(\tau \circ r_{|v_m|}) \\ &= \langle \widehat{N}_{m+1}(f \circ r),\ \widehat{N}_{m+1}(g \circ r) \rangle \\ &= K_{m+1}(f \circ r,\ g \circ r)\ , \end{aligned}$$

as desired.

Having proven that K_m is reversal symmetric for $1 \leq m \leq n$, we will now follow the general method of Case (1) to show that K_n is not reversal invariant. Let $1 \leq m \leq n-1$ and assume that K_m is not reversal invariant. Let $f \in \mathrm{Im}(v_m)$ be such that $K_m(f, f \circ r) < 1$. Then to prove that K_{m+1} is not reversal invariant, it suffices to show that either $K_{m+1}(e_f,\ e_f \circ r) < 1$, or there exists $f' \in \mathrm{Im}(v_m)$ such that $\lambda(f') > \lambda(f)$ and $K_m(f', f' \circ r) < 1$.

First suppose $K_m(f \circ r,\ e_f \circ h_i) < 1$ for all $2 \leq i \leq |v_{m+1}| - |v_m| + 1$. Then, noticing that

$$e_f \circ h_1 = f \quad \text{and} \quad e_f \circ r_{|v_{m+1}|} \circ \chi(h_1) = e_f \circ h_1 \circ r_{|v_m|} = f \circ r_{|v_m|}\ ,$$

by considering the template $\tau = \widehat{N}_m(f \circ r) \in \mathcal{T}_m$ we see that

$$N_{m+1}(e_f)(\tau) = \max_{h \in H_m} K_m(e_f \circ h,\ f \circ r) < 1\ ,$$

while

$$N_{m+1}(e_f \circ r)(\tau) = \max_{h \in H_m} K_m(e_f \circ r_{|v_{m+1}|} \circ h_i,\ f \circ r_{|v_m|}) = K_m(f \circ r,\ f \circ r) = 1\ .$$

This shows that $N_{m+1}(e_f) \neq N_{m+1}(e_f \circ r)$, and hence $K_{m+1}(e_f,\ e_f \circ r) < 1$ by Proposition 5.4.

On the other hand, suppose $K_m(f \circ r,\ e_f \circ h_i) = 1$ for some $2 \leq i \leq |v_{m+1}| - |v_m| + 1$. Using the reversal symmetry of K_m, we get

$$K_m(f,\ e_f \circ h_i \circ r) = K_m(f \circ r,\ e_f \circ h_i) = 1\ .$$

Then notice that we must have

$$K_m(e_f \circ h_i,\ e_f \circ h_i \circ r) < 1\ ,$$

for otherwise

$$N_m(f) = N_m(e_f \circ h_i \circ r) = N_m(e_f \circ h_i) = N_m(f \circ r)\ ,$$

contradicting the fact that $K_m(f, f \circ r) < 1$. Since e_f is constructed by growing the tail of f, we have $\lambda(e_f \circ h_i) \geq \lambda(f) + 1 > \lambda(f)$. Thus we can take the new witness f' to be $e_f \circ h_i \in \mathrm{Im}(v_m)$, and we are done. □

D.3 Equivalence Classes of the Derived Kernel

We first present a sequence of preliminary results that will help us prove Theorem 5.13. The following lemma tells us that if we have a string that is periodic with two different periods, then that string is actually periodic with a period that is the greatest common divisor of the two original periods.

Lemma D.4. *Let f be a p-string that is both k_1-periodic and k_2-periodic, and let $k = \gcd(k_1, k_2)$. If $p \geq k_1 + k_2$, then f is k-periodic.*

Proof. Without loss of generality we may assume $k_1 \le k_2$. Write $f = a_1 \dots a_p$. Since we already know f is k_1-periodic, to prove f is k-periodic it suffices to show that the substring $a_1 \dots a_{k_1}$ is k-periodic.

Fix $1 \le i \le k_1 - k$. Starting from a_i, we repeatedly use the periodicity of f to move from one element of f to the next by jumping either k_1 indices to the right or k_2 indices to the left. If we follow the rule of jumping to the left whenever possible, then we can reach a_{i+k} while staying within the string f. More formally, consider the function $j\colon \mathbb{N}_0 \to \mathbb{Z}$ given by $j(0) = 0$, and for $d \in \mathbb{N}$,

$$j(d) = \begin{cases} j(d-1) - k_2 & \text{if } j(d-1) - k_2 \ge -i+1, \\ j(d-1) + k_1 & \text{otherwise.} \end{cases}$$

We claim that $-i+1 \le j(d) \le k_1 + k_2 - i$ for all $d \in \mathbb{N}$, and that $j(d^*) = k$ for some $d^* \in \mathbb{N}$. The first claim tells us that $1 \le i + j(d) \le p$ for all $d \in \mathbb{N}$, and thus using the periodicity of f, we have $a_i = a_{i+j(d)}$ for all $d \in \mathbb{N}$. The second claim then gives us $a_i = a_{i+j(d^*)} = a_{i+k}$, as desired.

For the first claim, it is clear from the definition of j that $j(d) \ge -i+1$ for all $d \in \mathbb{N}$. Now suppose the contrary that $j(d) > k_1 + k_2 - i$ for some $d \in \mathbb{N}$; let d' be the smallest such d. We consider the move from $j(d'-1)$ to $j(d')$. On the one hand, if $j(d') = j(d'-1) - k_2$, then $j(d'-1) = j(d') + k_2 > k_1 + k_2 - i$, contradicting our choice of d'. On the other hand, if $j(d') = j(d'-1) + k_1$, then $j(d'-1) - k_2 = j(d') - k_1 - k_2 > -i$, which means we should have jumped to the left to go from $j(d'-1)$ to $j(d')$. This shows that $j(d) \le k_1 + k_2 - i$ for all $d \in \mathbb{N}$.

For the second claim, we first note that by the Euclidean algorithm we can find $x, y \in \mathbb{N}$ satisfying $k = xk_1 - yk_2$. Let $\widetilde{d}_1$ be the first time we jump to the right x times. That is, $\widetilde{d}_1$ is the smallest $d \in \mathbb{N}$ such that from $j(0)$ to $j(d)$ we have jumped to the right x times, not necessarily consecutively. Note that such a $\widetilde{d}_1$ must exist, as we cannot jump to the left all the time. Similarly, let $\widetilde{d}_2$ be the first time we jump to the left y times. Observe that $\widetilde{d}_1 \ne \widetilde{d}_2$. We now consider two cases.

1. If $\widetilde{d}_1 < \widetilde{d}_2$, then by the time we reach $j(\widetilde{d}_1)$ we would have jumped to the right x times and to the left $\widetilde{d}_1 - x < y$ times. Thus

   $$j(\widetilde{d}_1) = xk_1 - (\widetilde{d}_1 - x)k_2 = k + yk_2 - (\widetilde{d}_1 - x)k_2 = k + (x + y - \widetilde{d}_1)k_2 \ .$$

   Since $x + y - \widetilde{d}_1 > 0$, we can now jump to the left $x + y - \widetilde{d}_1$ times to obtain

   $$j(x+y) = j(\widetilde{d}_1) - (x + y - \widetilde{d}_1)k_2 = k \ ,$$

   as desired.

2. If $\widetilde{d}_1 > \widetilde{d}_2$, then by the time we reach $j(\widetilde{d}_2)$ we would have jumped to the left y times and to the right $\widetilde{d}_2 - y < x$ times. Thus

   $$j(\widetilde{d}_2) = (\widetilde{d}_2 - y)k_1 - yk_2 = (\widetilde{d}_2 - y)k_1 + k - xk_1 = k + (\widetilde{d}_2 - x - y)k_1 \ .$$

   Note that since $i \le k_1 - k \le k_2 - k$, for any $0 \le d \le x + y - \widetilde{d}_2 - 1$ we have

   $$j(\widetilde{d}_2) + dk_1 - k_2 \le k + (\widetilde{d}_2 - x - y)k_1 + (x + y - \widetilde{d}_2 - 1)k_1 - k_2 = k - k_1 - k_2 \le -i \ .$$

   This means from $j(\widetilde{d}_2)$, the next $x + y - \widetilde{d}_2$ moves have to be jumps to the right, and thus

   $$j(x+y) = j(\widetilde{d}_2) + (x + y - \widetilde{d}_2)k_1 = k \ ,$$

   as desired.

□

Now we present the following result, which constitutes a substantial part of the proof of Theorem 5.13. The following lemma says that when the jump in the patch sizes is larger than the jumps in the previous layer, we obtain a new equivalence class of the derived kernel.

Lemma D.5. *Consider an architecture with $n \geq 2$ layers. Let $q \in \mathbb{N}$ and $2 \leq m \leq n$, and suppose that: (i) $|v_m| - |v_{m-1}| = q$, (ii) $|v_m| \geq 4q + 2$, and (iii) at layer $m-1$ we have*

$$K_{m-1}(f, g) = 1 \quad \textit{implies} \quad f = g \text{ or } f \overset{(k)}{\sim} g \text{ for some } 2 \leq k \leq q+1 \ .$$

Then at layer m we have

$$K_m(f, g) = 1 \quad \textit{if and only if} \quad f = g \text{ or } f \overset{(k)}{\sim} g \text{ for some } 2 \leq k \leq q+1 \ .$$

Proof. We first prove the forward implication. Let $f = a_1 \dots a_{|v_m|}$ and $g = b_1 \dots b_{|v_m|}$ be two strings in $\mathrm{Im}(v_m)$ such that $K_m(f, g) = 1$. For $1 \leq i \leq q+1$, let $f_i = a_i \dots a_{i+|v_{m-1}|-1}$ and $g_i = b_i \dots b_{i+|v_{m-1}|-1}$ be the i-th $|v_{m-1}|$-substrings of f and g, respectively. We divide the proof into several steps.

Step 1. By Proposition 5.4, $K_m(f, g) = 1$ implies $N_m(f) = N_m(g)$, and so $N_m(f)(\tau) = N_m(g)(\tau)$ for all templates $\tau \in \mathcal{T}_{m-1}$. In particular, by taking $\tau = \widehat{N}_{m-1}(f_i)$, $1 \leq i \leq q+1$, we see that $N_m(g)(\tau) = N_m(f)(\tau) = 1$. This means there exists $1 \leq j \leq q+1$ such that $K_{m-1}(f_i, g_j) = 1$. Similarly, for each $1 \leq j \leq q+1$ there exists $1 \leq i \leq q+1$ such that $K_{m-1}(f_i, g_j) = 1$.

Step 2. We will now show that

$$f_1 = g_1 \quad \text{or} \quad f_1 \text{ is periodic with period } \leq q+1 \ .$$

Let j^* be the smallest index $1 \leq j \leq q+1$ such that $K_{m-1}(f_1, g_j) = 1$. If $f_1 \overset{(k)}{\sim} g_{j^*}$ for some $2 \leq k \leq q+1$, then f_1 is k-periodic and we are done. Now assume $f_1 = g_{j^*}$. If $j^* = 1$, then $f_1 = g_1$ and we are done. Suppose now that $j^* > 1$. By the choice of j^*, we can find $2 \leq i^* \leq q+1$ such that $K_{m-1}(f_{i^*}, g_{j^*-1}) = 1$. We consider two cases.

1. Case 1: $f_{i^*} = g_{j^*-1}$, which means $b_d = a_{d+i^*-j^*+1}$ for $j^* - 1 \leq d \leq j^* + |v_{m-1}| - 2$. Then for each $1 \leq d \leq |v_{m-1}| - i^*$, from $f_1 = g_{j^*}$ we have $a_d = b_{d+j^*-1}$, and from $f_{i^*} = g_{j^*-1}$ we have $b_{d+j^*-1} = a_{(d+j^*-1)+i^*-j^*+1} = a_{d+i^*}$. This shows that f_1 is i^*-periodic.

2. Case 2: $f_{i^*} \overset{(k)}{\sim} g_{j^*-1}$ for some $2 \leq k \leq q+1$. Since g_{j^*-1} is k-periodic, the substring $a_1 \dots a_{|v_{m-1}|-1} = b_{j^*} \dots b_{j^*+|v_{m-1}|-2}$ is also k-periodic. Moreover, since f_{i^*} is k-periodic, we also have $a_{|v_{m-1}|} = a_{|v_{m-1}|-k}$. This shows that $f_1 = a_1 \dots a_{|v_{m-1}|}$ is k-periodic.

Step 3. Similar to the previous step, we can show that the following conditions are true:

$$\begin{aligned} f_1 = g_1 \quad &\text{or} \quad g_1 \text{ is periodic with period } \leq q+1, \\ f_{q+1} = g_{q+1} \quad &\text{or} \quad f_{q+1} \text{ is periodic with period } \leq q+1, \quad \text{and} \\ f_{q+1} = g_{q+1} \quad &\text{or} \quad g_{q+1} \text{ is periodic with period } \leq q+1 \ . \end{aligned}$$

Thus we can obtain the following conclusion:

$$\begin{aligned} f_1 = g_1 \quad &\text{or} \quad f_1 \text{ and } g_1 \text{ are periodic with period } \leq q+1, \quad \text{and} \\ f_{q+1} = g_{q+1} \quad &\text{or} \quad f_{q+1} \text{ and } g_{q+1} \text{ are periodic with period } \leq q+1 \ . \end{aligned}$$

Note that in the statement above, when f_1 and g_1 are periodic their periods do not have to be the same, and similarly for f_{q+1} and g_{q+1}.

Step 4. Finally, we now show that either $f = g$ or $f \overset{(k)}{\sim} g$ for some $2 \leq k \leq q+1$. Using the conclusion of the previous step, we consider four possibilities.

1. Suppose $f_1 = g_1$ and $f_{q+1} = g_{q+1}$. In this case we immediately obtain $f = g$.

2. Suppose $f_1 = g_1$, f_{q+1} is k_1-periodic, and g_{q+1} is k_2-periodic, where $k_1, k_2 \leq q+1$. Then the substring $a_{q+1} \dots a_{|v_{m-1}|} = b_{q+1} \dots b_{|v_{m-1}|}$, which is of length $|v_{m-1}| - q = |v_m| - 2q \geq 2q+2$, is both k_1-periodic and k_2-periodic. By Lemma D.4, this implies that $a_{q+1} \dots a_{|v_{m-1}|} = b_{q+1} \dots b_{|v_{m-1}|}$ is periodic with period $k = \gcd(k_1, k_2)$. In particular, this means both f_{q+1} and g_{q+1} are also k-periodic. Now given $|v_{m-1}| + 1 \leq d \leq |v_m|$, choose $x \in \mathbb{N}$ such that $q+1 \leq d - xk \leq |v_{m-1}|$, and observe that $a_d = a_{d-xk} = b_{d-xk} = b_d$. Together with $f_1 = g_1$, we conclude that $f = g$.

3. Suppose f_1 and g_1 are periodic with period at most $q+1$, and $f_{q+1} = g_{q+1}$. By the same argument as in the previous case, we can show that $f = g$.

4. Now suppose that f_1, f_{q+1}, g_1, and g_{q+1} are periodic with periods at most $q+1$ (but the periods do not have to be the same). Applying Lemma D.4 to the substring $a_{q+1} \dots a_{|v_{m-1}|}$, we see that f_1 and f_{q+1} are k_1-periodic for some $k_1 \leq q+1$, and hence f is also k_1-periodic. Similarly, g is k_2-periodic for some $k_2 \leq q+1$. Let $1 \leq j \leq q+1$ be such that $K_{m-1}(f_1, g_j) = 1$. We have two possibilities to consider.

   (a) If $f_1 = g_j$, then by Lemma D.4 we know that $f_1 = g_j$ is periodic with period $k' = \gcd(k_1, k_2) \leq q+1$, and thus f and g are also k'-periodic. Since $f_1 = g_j$, we conclude that $f \overset{(k')}{\sim} g$.

   (b) On the other hand, suppose $f_1 \overset{(k)}{\sim} g_j$ for some $2 \leq k \leq q+1$. Since f_1 is both k-periodic and k_1-periodic, Lemma D.4 tells us that f_1 is periodic with period $k_3 = \gcd(k, k_1)$, and hence f is also k_3-periodic. Similarly, since g_j is k-periodic and k_2-periodic, Lemma D.4 tells us that g_j is periodic with period $k_4 = \gcd(k, k_2)$, and hence g is k_4-periodic. In particular, this means both f and g are k-periodic. Then from $f_1 \overset{(k)}{\sim} g_j$, we conclude that $f \overset{(k)}{\sim} g$.

This completes the proof of the forward direction.

Now we show the converse direction. Clearly if $f = g$ then we have $K_m(f, g) = 1$. Now suppose $f \overset{(k)}{\sim} g$ for some $2 \leq k \leq q+1$. This implies that the collection of $|v_{m-1}|$-substrings of f contains the same unique substrings as the collection of $|v_{m-1}|$-substrings of g. That is, as sets, $\{f_i \mid 1 \leq i \leq q+1\} = \{g_i \mid 1 \leq i \leq q+1\}$. Then for all $\tau \in \mathcal{T}_{m-1}$,

$$N_m(f)(\tau) = \max_{1 \leq i \leq q+1} \langle \widehat{N}_{m-1}(f_i),\, \tau \rangle = \max_{1 \leq i \leq q+1} \langle \widehat{N}_{m-1}(g_i),\, \tau \rangle = N_m(g)(\tau) \ .$$

Therefore, $N_m(f) = N_m(g)$, and we conclude that $K_m(f, g) = 1$, as desired. □

The following lemma, which is very similar to Lemma D.5, states that if at some layer we see a jump in the patch sizes that we have encountered before, then we do not obtain new equivalence classes of the derived kernel.

Lemma D.6. *Consider an architecture with $n \geq 2$ layers. Let $q \in \mathbb{N}$ and $2 \leq m \leq n$, and suppose that: (i) $|v_m| - |v_{m-1}| \leq q$, (ii) $|v_m| \geq 4q+2$, and (iii) at layer $m-1$ we have*

$$K_{m-1}(f, g) = 1 \quad \text{if and only if} \quad f = g \text{ or } f \overset{(k)}{\sim} g \text{ for some } 2 \leq k \leq q+1 \ .$$

Then at layer m we have

$$K_m(f, g) = 1 \quad \text{if and only if} \quad f = g \text{ or } f \overset{(k)}{\sim} g \text{ for some } 2 \leq k \leq q+1 \ .$$

Proof. The proof of the forward direction is identical to the proof of Lemma D.5. For the converse direction, clearly $f = g$ implies $K_m(f, g) = 1$. Now suppose $f \overset{(k)}{\sim} g$ for some $2 \leq k \leq q+1$. Let $d = |v_m| - |v_{m-1}| \leq q$. Then for each $1 \leq i \leq d+1$ we have $f_i \overset{(k)}{\sim} g_i$, where f_i and g_i are the i-th

$|v_{m-1}|$-substrings of f and g, respectively. By our assumption, this means $K_{m-1}(f_i, g_i) = 1$, and thus $\widehat{N}_{m-1}(f_i) = \widehat{N}_{m-1}(g_i)$. Then for all $\tau \in \mathcal{T}_{m-1}$,

$$N_m(f)(\tau) = \max_{1 \le i \le q+1} \langle \widehat{N}_{m-1}(f_i),\, \tau \rangle = \max_{1 \le i \le q+1} \langle \widehat{N}_{m-1}(g_i),\, \tau \rangle = N_m(g)(\tau) \ .$$

Hence $N_m(f) = N_m(g)$, and we conclude that $K_m(f, g) = 1$, as desired. □

Given the preliminary results above, we can now prove Theorem 5.13 easily.

Proof of Theorem 5.13. Let $\ell_1 = 0$, and for $2 \le m \le n$ let ℓ_m denote the maximum jump in the subsequent patch sizes up to layer m,

$$\ell_m = \max_{2 \le m' \le m} \big(|v_m| - |v_{m-1}| \big) \ .$$

Note that $\ell_m \le \ell_{m+1}$ and $\ell_n = \ell$. We will show that for each $1 \le m \le n$,

$$K_m(f, g) = 1 \quad \text{ if and only if } \quad f = g \text{ or } f \overset{(k)}{\sim} g \text{ for some } 2 \le k \le \ell_m + 1 \ .$$

The statement of the theorem will then follow from the claim above by taking $m = n$.

We proceed by induction. The claim above is true for $m = 1$ since we assume K_1 is fully discriminative. Now assume the claim is true at layer $m - 1$. Note that at layer m we have

$$|v_m| \ge |v_1| + \ell_m \ge 3\ell + 2 + \ell_m \ge 4\ell_m + 2 \ .$$

If $\ell_m > \ell_{m-1}$, then $|v_m| - |v_{m-1}| = \ell_m$. By the induction hypothesis, at layer $m - 1$ we have

$$K_{m-1}(f, g) = 1 \quad \text{ implies } \quad f = g \text{ or } f \overset{(k)}{\sim} g \text{ for some } 2 \le k \le \ell_m + 1 \ ,$$

and so by Lemma D.5 we conclude that the claim holds at layer m. On the other hand, if $\ell_m = \ell_{m-1}$, then by the induction hypothesis at layer $m - 1$ we have

$$K_{m-1}(f, g) = 1 \quad \text{ if and only if } \quad f = g \text{ or } f \overset{(k)}{\sim} g \text{ for some } 2 \le k \le \ell_m + 1 \ ,$$

and so by Lemma D.6 we conclude that the claim holds at layer m. □

D.4 Mirror Symmetry in Two-Dimensional Images

The proof of Proposition 5.16 is identical to the proof of Proposition 5.14, but for completeness, we provide the detail below.

Proof of Proposition 5.16. Let $\chi \colon H_m \to H_m$ be the function

$$\chi(h_{i,j}) = h_{|v_{m+1}| - |v_i| + 2 - i,\, j} \quad \text{ for } 1 \le i, j \le |v_{m+1}| - |v_m| + 1 \ .$$

Note that χ is a bijection, $\chi(\chi(h)) = h$, and by construction it has the property that

$$r_{|v_{m+1}|} \circ \chi(h) = h \circ r_{|v_m|} \quad \text{ for } h \in H_m \ .$$

Let $f \in \mathrm{Im}(v_{m+1})$, and suppose we are given $\tau = \widehat{N}_m(g') \in \mathcal{T}_m$ with the property that $\tau \circ r = \widehat{N}_m(g' \circ r) \in \mathcal{T}_m$. Then, using the assumption that K_m is reflection symmetric,

$$N_{m+1}(f)(\tau) = \max_{h \in H_m} \ K_m(f \circ h,\, g')$$

$$
\begin{aligned}
&= \max_{h \in H_m} \; K_m(f \circ h \circ r_{|v_m|},\, g' \circ r_{|v_m|}) \\
&= \max_{h \in H_m} \; K_m(f \circ r_{|v_{m+1}|} \circ \chi(h),\, g' \circ r_{|v_m|}) \\
&= N_{m+1}(f \circ r_{|v_{m+1}|})(\tau \circ r_{|v_m|}) \;\; .
\end{aligned}
$$

Now suppose $\mathcal{T}_m$ is closed under the reflection operation, which means $\tau \in \mathcal{T}_m$ implies $\tau \circ r \in \mathcal{T}_m$. Noting that the mapping

$$
\mathcal{T}_m \ni \tau = \widehat{N}_m(g') \stackrel{r}{\longmapsto} \widehat{N}_m(g' \circ r) = \tau \circ r \in \mathcal{T}_m
$$

is a bijection, the previous identity then gives us

$$
\|N_{m+1}(f)\| = \|N_{m+1}(f \circ r)\| \;\; ,
$$

and therefore,

$$
\widehat{N}_{m+1}(f)(\tau) = \frac{N_{m+1}(f)(\tau)}{\|N_{m+1}(f)\|} = \frac{N_{m+1}(f \circ r_{|v_{m+1}|})(\tau \circ r_{|v_m|})}{\|N_{m+1}(f \circ r_{|v_{m+1}|})\|} = \widehat{N}_{m+1}(f \circ r_{|v_{m+1}|})(\tau \circ r_{|v_m|}) \;\; .
$$

Hence, for all $f, g \in \mathrm{Im}(v_{m+1})$ we have

$$
\begin{aligned}
K_{m+1}(f,g) &= \big\langle \widehat{N}_{m+1}(f),\, \widehat{N}_{m+1}(g) \big\rangle \\
&= \sum_{\tau \in \mathcal{T}_m} \widehat{N}_{m+1}(f)(\tau)\; \widehat{N}_{m+1}(g)(\tau) \\
&= \sum_{\tau \in \mathcal{T}_m} \widehat{N}_{m+1}(f \circ r_{|v_{m+1}|})(\tau \circ r_{|v_m|})\; \widehat{N}_{m+1}(g \circ r_{|v_{m+1}|})(\tau \circ r_{|v_m|}) \\
&= \big\langle \widehat{N}_{m+1}(f \circ r),\, \widehat{N}_{m+1}(g \circ r) \big\rangle \\
&= K_{m+1}(f \circ r,\, g \circ r) \;\; ,
\end{aligned}
$$

as desired. □

References

[1] Nachman Aronszajn. Theory of reproducing kernels. *Transactions of the American Mathematical Society*, 68:337–404, 1950.

[2] Gleb Beliakov, Ana Pradera, and Tomasa Calvo. *Aggregation Functions: A Guide for Practitioners*. Springer, 2007.

[3] Yoshua Bengio. Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2(1):1–127, 2009.

[4] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In *Advances in Neural Information Processing Systems 19 (NIPS'06)*. MIT Press, 2006.

[5] Yoshua Bengio and Yann LeCun. Scaling learning algorithms towards AI. In *Large-Scale Kernel Machines*. MIT Press, 2007.

[6] James Bergstra, Guillaume Desjardins, Pascal Lamblin, and Yoshua Bengio. Quadratic polynomials learn better image features. Technical Report 1337, Département d'Informatique et de Recherche Opérationnelle, Université de Montréal, 2009.

[7] Y-Lan Boureau, Francis Bach, Yann LeCun, and Jean Ponce. Learning mid-level features for recognition. In *Proceedings of the International Conference on Computer Vision and Pattern Recognition (CVPR'10)*. IEEE, 2010.

[8] Jake Bouvrie. *Hierarchical Learning: Theory with Applications in Speech and Vision*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, 2009.

[9] Jake Bouvrie, Tony Ezzat, and Tomaso Poggio. Localized spectro-temporal cepstral analysis of speech. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP'08)*, 2008.

[10] Jake Bouvrie, Lorenzo Rosasco, and Tomaso Poggio. On invariance in hierarchical models. In *Advances in Neural Information Processing Systems 22 (NIPS'09)*. MIT Press, 2009.

[11] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: deep neural networks with multitask learning. In *Proceedings of the 25th International Conference on Machine Learning (ICML'08)*. ACM, 2008.

[12] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[13] Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, 11:625–660, 2010.

[14] Theodoros Evgeniou, Massimiliano Pontil, and Tomaso Poggio. Regularization networks and support vector machines. *Advances in Computational Mathematics*, 13(1):1–50, 2000.

[15] Tony Ezzat, Jake Bouvrie, and Tomaso Poggio. AM-FM demodulation of spectrograms using localized 2D max-Gabor analysis. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP'07)*. IEEE, 2007.

[16] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4):193–202, 1980.

[17] Martin Giese and Tomaso Poggio. Neural mechanisms for the recognition of biological movements. *Nature Reviews Neuroscience*, 4:179–192, 2003.

[18] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS'10)*. Society for Artificial Intelligence and Statistics, 2010.

[19] Ian Goodfellow, Quoc V. Le, Andrew Saxe, Honglak Lee, and Andrew Y. Ng. Measuring invariances in deep networks. In *Advances in Neural Information Processing Systems 22 (NIPS'09)*. MIT Press, 2009.

[20] Raia Hadsell, Pierre Sermanet, Marco Scoffier, Ayse Erkan, Koray Kavackuoglu, Urs Muller, and Yann LeCun. Learning long-range vision for autonomous off-road driving. *Journal of Field Robotics*, 26(2):120–144, February 2009.

[21] Bernd Heisele, Thomas Serre, Massimiliano Pontil, Thomas Vetter, and Tomaso Poggio. Categorization by learning and combining object parts. In *Advances in Neural Information Processing Systems 14 (NIPS'01)*. MIT Press, 2001.

[22] Geoffrey Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006.

[23] Geoffrey Hinton and Ruslan Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, July 2006.

[24] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[25] David Hubel and Torsten Wiesel. Receptive fields and functional architecture of monkey striate cortex. *The Journal of Physiology*, 195(1):215–243, 1968.

[26] Kevin Jarrett, Koray Kavukcuoglu, Marc'Aurelio Ranzato, and Yann LeCun. What is the best multi-stage architecture for object recognition? In *Proceedings of the International Conference on Computer Vision (ICCV'09)*. IEEE, 2009.

[27] Hueihan Jhuang, Thomas Serre, Lior Wolf, and Tomaso Poggio. A biologically inspired system for action recognition. In *Proceedings of the International Conference on Computer Vision (ICCV'07)*. IEEE, 2007.

[28] Koray Kavukcuoglu, Marc'Aurelio Ranzato, Rob Fergus, and Yann LeCun. Learning invariant features through topographic filter maps. In *Proceedings of the International Conference on Computer Vision and Pattern Recognition (CVPR'09)*. IEEE, 2009.

[29] Hugo Larochelle, Dumitru Erhan, Aaron Courville, James Bergstra, and Yoshua Bengio. An empirical evaluation of deep architectures on problems with many factors of variation. In *Proceedings of the 24th International Conference on Machine Learning (ICML'07)*. ACM, 2007.

[30] Nicolas Le Roux and Yoshua Bengio. Deep belief networks are compact universal approximators. *Neural Computation (to appear)*, 2010.

[31] Yann LeCun, Bernard Boser, John Denker, Donnie Henderson, Richard Howard, Wayne Hubbard, and Larry Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989.

[32] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998.

[33] Yann LeCun, Fu-Jie Huang, and Leon Bottou. Learning methods for generic object recognition with invariance to pose and lighting. In *Proceedings of the International Conference on Computer Vision and Pattern Recognition (CVPR'04)*. IEEE, 2004.

[34] Yann LeCun, Urs Muller, Jan Ben, Eric Cosatto, and Beat Flepp. Off-road obstacle avoidance through end-to-end learning. In *Advances in Neural Information Processing Systems 18 (NIPS'05)*. MIT Press, 2005.

[35] Daniel Lee and H. Sebastian Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401(6755):788–791, October 1999.

[36] Honglak Lee, Alexis Battle, Rajat Raina, and Andrew Y. Ng. Efficient sparse coding algorithms. In *Advances in Neural Information Processing Systems 19 (NIPS'06)*. MIT Press, 2006.

[37] Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y. Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th International Conference on Machine Learning (ICML'09)*. ACM, 2009.

[38] Honglak Lee, Peter Pham, Yan Largman, and Andrew Y. Ng. Unsupervised feature learning for audio classification using convolutional deep belief networks. In *Advances in Neural Information Processing Systems 22 (NIPS'09)*. MIT Press, 2009.

[39] Julien Mairal, Francis Bach, Jean Ponce, Guillermo Sapiro, and Andrew Zisserman. Discriminative learned dictionaries for local image analysis. In *Proceedings of the International Conference on Computer Vision and Pattern Recognition (CVPR'08)*. IEEE, 2008.

[40] Piotr Mirowski, Yann LeCun, Deepak Madhavan, and Ruben Kuzniecky. Comparing SVM and convolutional networks for epileptic seizure prediction from intracranial EEG. In *Proceedings of the IEEE International Workshop on Machine Learning and Signal Processing (MLSP'08)*. IEEE, 2008.

[41] Piotr Mirowski, Deepak Madhavan, Yann LeCun, and Ruben Kuzniecky. Classification of patterns of EEG synchronization for seizure prediction. *Clinical Neurophysiology*, 120(11):1927–1940, November 2009.

[42] Andriy Mnih and Geoffrey Hinton. A scalable hierarchical distributed language model. In *Advances in Neural Information Processing Systems 21 (NIPS'08)*. MIT Press, 2008.

[43] Jim Mutch, Ulf Knoblich, and Tomaso Poggio. CNS: A GPU-based framework for simulating cortically-organized networks. Technical Report MIT-CSAIL-TR-2010-013, Massachusetts Institute of Technology, 2010.

[44] Jim Mutch and David Lowe. Multiclass object recognition with sparse, localized features. In *Proceedings of the International Conference on Computer Vision and Pattern Recognition (CVPR'06)*. IEEE, 2006.

[45] Nicolas Pinto, David Doukhan, James DiCarlo, and David Cox. A high-throughput screening approach to discovering good forms of biologically inspired visual representation. *PLoS Computational Biology*, 5(11):e1000579, November 2009.

[46] Tomaso Poggio and Steve Smale. The mathematics of learning: Dealing with data. *Notices of the American Mathematical Society*, 50(5):537–544, 2003.

[47] Rajat Raina, Anand Madhavan, and Andrew Y. Ng. Large-scale deep unsupervised learning using graphics processors. In *Proceedings of the 26th International Conference on Machine Learning (ICML'09)*. ACM, 2009.

[48] Marc'Aurelio Ranzato, Christopher Poultney, Sumit Chopra, and Yann LeCun. Efficient learning of sparse representations with an energy-based model. In *Advances in Neural Information Processing Systems 19 (NIPS'06)*. MIT Press, 2006.

[49] Fred Rieke, David Warland, Rob de Ruyter van Steveninck, and William Bialek. *Spikes: Exploring the Neural Code*. MIT Press, 1999.

[50] Maximilian Riesenhuber and Tomaso Poggio. Hierarchical models of object recognition in cortex. *Nature Neuroscience*, 2(11):1019–1025, November 1999.

[51] Raúl Rojas. Networks of width one are universal classifiers. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN'03)*, 2003.

[52] Ruslan Salakhutdinov and Geoffrey Hinton. Learning a nonlinear embedding by preserving class neighbourhood structure. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS'07)*. Society for Artificial Intelligence and Statistics, 2007.

[53] Bernhard Schölkopf, Christopher J. C. Burges, and Alexander J. Smola, editors. *Advances in Kernel Methods: Support Vector Learning*. MIT Press, Cambridge, MA, USA, 1999.

[54] Thomas Serre, Aude Oliva, and Tomaso Poggio. A feedforward architecture accounts for rapid categorization. *Proceedings of the National Academy of Sciences*, 104(15):6424–6429, 2007.

[55] Thomas Serre, Lior Wolf, and Tomaso Poggio. Object recognition with features inspired by visual cortex. In *Proceedings of the International Conference on Computer Vision and Pattern Recognition (CVPR'05)*. IEEE, 2005.

[56] Steve Smale, Lorenzo Rosasco, Jake Bouvrie, Andrea Caponnetto, and Tomaso Poggio. Mathematics of the neural response. *Foundations of Computational Mathematics*, 10(1):67–91, 2010.

[57] Ilya Sutskever and Geoffrey Hinton. Deep, narrow sigmoid belief networks are universal approximators. *Neural Computation*, 20(11):2629–2636, 2008.

[58] Srinivas Turaga, Joseph Murray, Viren Jain, Fabian Roth, Moritz Helmstaedter, Kevin Briggman, Winfried Denk, and H. Sebastian Seung. Convolutional networks can learn to generate affinity graphs for image segmentation. *Neural Computation*, 22(2):511–538, 2010.

[59] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th International Conference on Machine Learning (ICML'08)*. ACM, 2008.

[60] Grace Wahba. *Spline Models for Observational Data*. Society for Industrial and Applied Mathematics, 1990.

[61] Andre Wibisono. Generalization and properties of the neural response. Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, 2010.

[62] Jianchao Yang, Kai Yu, Yihong Gong, and Thomas Huang. Linear spatial pyramid matching using sparse coding for image classification. In *Proceedings of the International Conference on Computer Vision and Pattern Recognition (CVPR'09)*. IEEE, 2009.

[63] Guoshen Yu and Jean-Jacques Slotine. Fast wavelet-based visual classification. In *Proceedings of the International Conference on Pattern Recognition (ICPR'08)*. IEEE, 2008.