Benefits of Max Pooling in Neural Networks: Theoretical and Experimental Evidence

Anonymous authors
Paper under double-blind review

Abstract

When deep neural networks became state of the art image classifiers, numerous max pooling operations were an important component of the architecture. However, modern computer vision networks typically have few if any max pooling operations. To understand whether this trend is justified, we develop a mathematical framework analyzing ReLU based approximations of max pooling, and prove a sense in which max pooling cannot be replicated. We formulate and analyze a class of optimal approximations, and find that residual can be made exponentially small in the kernel size, but only with an exponentially wide approximation.

This work gives a theoretical basis for understanding the reduced use of max pooling in newer architectures. Since max pooling does not seem necessary, we conclude that empirically the inputs on which max pooling is distinct – those with a large difference between the max and other values –is not a pattern prevalent in natural images.

1 Introduction

When convolutional neural networks first became state of the art image classifiers, max pooling was a fundamental aspect of modelling. More recent work has argued that max pooling operations are not necessary because strided convolutions composed with ReLU nonlinearity is simpler and more flexible (Springenberg et al. (2015)). Practice has followed this observation: early models such as VGG (Simonyan & Zisserman (2015)) and AlexNet (Krizhevsky et al. (2017)) featured several max pooling layers, but more recent models such as ResNets (He et al. (2016)) feature only a single max pooling layer, and some – such as InceptionV3 (Szegedy et al. (2016)) and mobilenetV3 (Howard et al. (2019)) – have none at all.[1]

This paper examines whether max pooling is a remnant of an earlier era when image classifiers were motivated by the visual cortex. We find that for some inputs, max pooling constructs very different features than an approximation by ReLU nonlinearity, thus it expresses a different inductive bias and can be appropriate in some situations. We derive comprehensive bounds on the error realized by approximating max functions with the composition of ReLU and linear operations, and find that a simple divide and conquer algorithm cannot be improved upon. We show that evaluating accurate approximations must be computationally complex, and present a mathematically interesting and efficient method for constructing (though not evaluating) these approximxations.

Our result does not imply that omitting max pooling from image classifiers is wrong. Rather, it says when it *could be wrong*. Max pooling will be strictly more expressive than a ReLU-based approximation on inputs with a large difference between the maximum and other values within pools.

Practice finds that max pooling is adequately replaced by ReLU nonlinearity and strided convolution, thus we infer that this characteristic is not empirically common in natural images.

Our main contributions are (1) introducing a novel generalization of max pooling (Section 4), (2) proving that the max function cannot be represented by simpler elements of this function class (Theorem 4), and

[1] All statements about historical models refer to their implementation in Torchvision (Marcel & Rodriguez (2010)), described at `https://pytorch.org/vision/stable/models.html`.

(3) analyzing experimentally the size and quality of approximations (Section 5). Other contributions are (1) formulating a sensible notion of separation for max pooling (Theorem 1, Theorem 2), and (2) connecting mathematically the average of subpool maximums with order statistics (Theorem 3).

2 Related work

The starting point for our study is the folk wisdom, demonstrated empirically in Springenberg et al. (2015), that strided convolution composed with nonlinearity, is preferable to max pooling because linear-ReLU blocks can fit a max pooling mapping if appropriate, and can seamlessly fit another functional form if not.

At the same time, much early work was concerned with analyzing the neurological basis of deep learning models, and actual analysis of mammal's eyes and brains have shown max pooling to be a biologically plausible operation (cf. for example Fukushima (1980); Riesenhuber & Poggio (1999)). Thus, we want to examine whether (1) max pooling can be efficiently simplified to linear operations and ReLU nonlinearity, and if not, (2) what would need to be true of data in order for this difference to be empirically unnecessary.

Boureau et al. (2010) is an early work comparing average and max pooling from an average-case statistical perspective. Like us, they also identify the dimensionality of the input as a key quantity. Despite a quite different method of analysis, they reach a complimentary conclusion. In their framework, sparse or low-probability features correspond to the corners of the input domain in our analysis. However, their result is notably weaker in comparing against only average pooling.

Serra et al. (2018), Arora et al. (2018), Bartlett et al. (2019), and Hanin & Rolnick (2019) have proposed and studied notions of the relative complexity of piecewise linear deep neural networks, but none examine specifically max pooling. Possibly, this is because it is mistakenly believed max pooling can be built up from a small number of ReLU layers. Most nonlinearities, xor example hard tanh (Collobert, 2004) $= x \mapsto \text{ReLU}(x+1) - 1 - \text{ReLU}(x-1)$, leaky ReLU (Maas et al., 2013)) $= x \mapsto \text{ReLU}(x) - .01 \times \text{ReLU}(-x)$, ReLU6 (Krizhevsky, 2012) $= x \mapsto \text{ReLU}(x) - \text{ReLU}(x-6)$, and hard sigmoid (Courbariaux et al., 2015) $= x \mapsto (\text{ReLU}(x+3) - \text{ReLU}(x-3))/6$, can, so this is understandable. However, one of our main results is to show that this is not true of max pooling. Our work does not apply to many other sources of nonlinearity, such as attention mechanisms.

Hertrich et al. (2021) is closely related to the theoretical component of our work. This study addresses similar aspects of ReLU-based approximations to the max function, and proves a technical sense in which max pooling is more expressive than just ReLUs. However, their proof technique relies crucially upon the discontinuity at zero, a dynamic not shared by our analysis, which is restricted to nonnegative inputs. We give precise error bounds, and are able to compare the errors of a large parameterized family of approximations with considerable precision to offer a more precise grasp of the practical tradeoff that applied approximations entail. Table 1 summarizes several results on the approximation of the maximum function by linear-ReLUs blocks.

Grüning & Barth (2022) find that *min* pooling can also be a useful pooling method. This is evidence that supports and is rationalized by our analysis in terms of the quantiles of the input to the pooling method. Essentially, if the true data is strongly determined by the nonlinear behavior of quantiles, then ReLU-based approaches are relatively disadvantaged.

3 The complexity of max pooling operations

In this section, we prove that in a simplified model, max pooling *requires* depth – multiple layers of ReLU nonlinearity are needed to effect the same computation, and more layers are needed for larger kernel sizes.

3.1 Simplifications

In the design of deep learning architectures, max pooling reduces dimensionality by summarizing the values of spatially nearby inputs. As a simplified mathematical model, we examine the approximation of a max *function*, putting aside "pooling"-specific considerations like stride, padding, and dilation which are

Depth	Features	Result
$\lceil \log_2 d \rceil$	small width	exact trivial Theorem 2
1	width $\uparrow \infty$	approximation possible (Hornik, 1991; Cybenko, 1989)
$\lceil \log_2 d \rceil - 1$	width $\uparrow \infty$	exact impossible on $\mathbb{R}^d$ (Hertrich et al., 2021)
$\lceil \log_2 d \rceil - 1$	$\mathcal{M}_d(R)$	exact impossible on $[0,1]^d$ (Theorem 4)
$\lceil \log_2 d \rceil - 1$	width $\uparrow \infty$	approximation experimentally difficult on $[0,1]^d$ (Section 5)

Table 1: A taxonomy of approximations to the max function by linear-ReLUs blocks in d dimensions. $\mathcal{M}_d(R)$ is our function class introduced in Section 4. The quality of the approximation depends crucially on depth, and as far as we are aware, this is the first paper to examine the finite width case.

ultimately linear pre-and post-processing. Similarly we treat as a general linear function any operator that amounts to a linear transformation at inference time, such as batch normalization, convolution, reindexing, or average pooling. Finally, we discuss only ReLU nonlinearities. We term these networks *purely feedforward ReLU* networks. Purely feedforward ReLU networks concisely reduce all aspects of the architecture to just the number of layers and their widths.

The phrase "order" indicates the size of the argument to a function. Thus, for example, the maximum over a 3×3 kernel is an order 9 max function.

3.2 Max pooling as a feature-builder

Telgarsky (2016) showed that deep neural networks cannot be concisely simulated by shallow networks. Their approach is to demonstrate a classification problem that is easy for deep networks, but is provably difficult for shallow networks. We do similarly by building a test problem on which max pooling succeeds and ReLU fails. However, Theorem 1 shows that for any dimensionality, a narrow purely feedforward network with a single source of nonlinearity can emit the same output as max pooling. Thus, prediction accuracy is not the correct metric by which to compare nonlinearities.

Theorem 1. *There exists a purely feedforward ReLU network $f : \mathbb{R}^d \to \mathbb{R}$ with d hidden neurons such that for all $\xi \in \mathbb{R}$, $f(x - \xi) \leq 0 \iff \max\{x_1, \ldots, x_d\} \leq \xi$.*

Proof.

$$\max\{x_1, \ldots, x_d\} \leq \xi \iff x_1 \leq \xi \text{ and } \ldots \text{ and } x_d \leq \xi \iff \sum_{k=1}^{d} \mathrm{ReLU}(x_k - \xi) \leq 0.$$

□

Max pooling is used in the construction of intermediate layer features, and not directly in the computation of final logits. Thus, in what follows we examine the ability of a purely feedforward ReLU network to achieve the same real-valued output as a max pooling operation with L_∞ error. And since diminishing the representation capacity of a single neuron necessarily reduces the expressivity of an entire network, we focus on a single neuron for simplicity.

3.3 Computing max using ReLU

Theorem 1 shows a positive result on the complexity of functions that composition of linear and ReLU functions can represent. In this section we begin developing our negative results.

The maximum of two values can be computed using the relationship

$$\max(a, b) = (\mathrm{ReLU}(a - b) + \mathrm{ReLU}(b - a) + a + b)/2. \tag{1}$$

There appears to be no tractable generalization of this formula for dimension $d > 2$. For example, in Appendix A, we give the analogous equation for the ternary case, and see that it is not linear in ReLU features. Building upon this, the appendix contains a heuristic argument that expressing the maximum of more than four variables without function composition may not be possible in general.

Nevertheless, Theorem 2 shows how with additional depth a purely feedfoward ReLU network can compute the maximum of many variables by recursively forming pairwise maxes.

Theorem 2. max: $\mathbb{R}^d \to \mathbb{R}$ *can be written as a* $\lceil \log_2(d) \rceil$*-hidden layer purely feedforward ReLU network, where the* k*th hidden layer has width* $2^{2+\lceil \log_2(d) \rceil - k}$.

Sketch of Proof. A variant of Equation 1 that is applicable to purely feedforward ReLU networks is

$$\begin{aligned} \max(x, y) &= (g \circ \mathrm{ReLU} \circ f)(x, y) \text{ where} \\ f(x, y) &= \begin{pmatrix} +x - y \\ -x + y \\ +x + y \\ -x - y \end{pmatrix} \\ g(x) &= (x_{0:n/4} + x_{n/4:n/2} + x_{n/2:3n/4} - x_{3n/4:n})/2. \end{aligned} \tag{2}$$

Here $x_{n_1:n_2}$ means the n_1th through the $(n_2 - 1)$th elements of x (inclusive), and n is the dimension of the input. f and g are linear. At the cost of quadrupling every layer width, ReLU can evaluate pairwise maxes and $\lceil \log_2(d) \rceil$ iterations of pairwise maxima can compute the maximum of d variables.

□

As an example, the max of five variables is simply written as three iterations of pairwise maxes:

$$\begin{aligned} \max(x_1, x_2, x_3, x_4, x_5) &= \max(z_1, z_2) \\ \text{where } z_1 &= \max(z_3, z_4), z_2 = \max(z_5, z_6) \\ \text{where } z_3 &= \max(x_1, x_2), z_4 = \max(x_2, x_3), z_5 = \max(x_3, x_4), z_6 = \max(x_4, x_5). \end{aligned}$$

Theorem 2 is an upper bound on the width and depth necessary to evaluate a max function. Corresponding lower bounds are more intricate. Table 1 outlines various possible converses, and the next section discusses the function class that characterizes our innovation.

4 The class of subpool max averages, $\mathcal{M}_d(R)$

Our work pertains to a particular function class, $\mathcal{M}_d(R)$. In subsection 4.1 we describe $\mathcal{M}_d(R)$, and in Section 5 we justify the relevance of this function class to deep learning.

4.1 Subpool maxes

For a vector $x \in \mathbb{R}^d$ and index set $J \subseteq \{1, \ldots, d\}$ we term $\max\{x_j : j \in J\}$ the J-subpool max of x. In words: subset the vector to those indices in J, then take the maximum element. For example, if $x = (3, 2, 10, 5)$, then the $\{1, 2, 4\}$-subpool max of x is $\max(3, 2, 5) = 5$. J-subpool maxes are generalizations of the max function that trade off complexity and accuracy in the sense that for $J_1 \subseteq J_2$, the J_1-subset max but simpler to compute than the J_2-subset max, but the error of a J_1-subpool max will always be at least that of a J_2-subpool max.

Let $C(j, r, d)$ denote the jth (out of $\binom{r}{d}$) subset of $\{1, \ldots, d\}$ of size r in the lexicographic ordering. For example, $C(1, 2, 3) = \{1, 2\}, C(2, 2, 3) = \{1, 3\}$ and $C(3, 2, 3) = \{2, 3\}$. To keep the notation manageable, let $\omega_{jr}(x)$ denote the $C(j, r, d)$-subpool max of x (the dimension d can be inferred from the size of x). And in order to more elegantly model a constant intercept, let the $C(1, 0, d)$-subpool max be $= 1$.

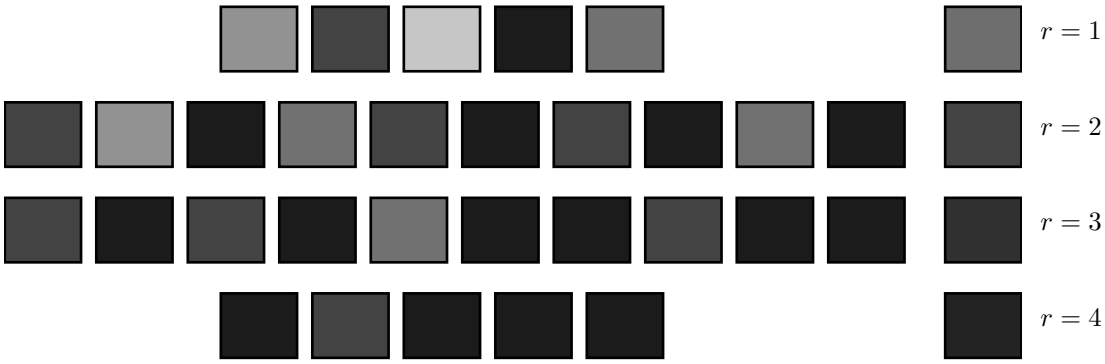


Figure 1: Averaged subpool maxes for $d = 5$. The numerical value of each node is presented as inverted grayscale. Each row $r = 1, 2, 3, 4$ indicates the averaged subpool maxes over pools of order r. As the order of the subpool maxes grows, the average value (on the right) grows darker towards the actual max.

Linear combinations of J-subpool maxes are a natural class of estimators to the max function. We organize subsets J by (1) the size of the subset, r, and (2) the subset index of that size, j. For $R \subseteq \{0, 1, \ldots, d-1, d\}$ let

$$\mathcal{M}_d(R) = \left\{ x \mapsto \sum_{r \in R} \sum_{j=1}^{\binom{d}{r}} \beta_r^j \omega_{jr}(x) : \beta_r^j \in \mathbb{R} \right\} \tag{3}$$

be the set of all linear combinations of r-subpool maxes, $r \in R$. Let a general element of $\mathcal{M}_d(R)$ be called an R-estimator. Theorem 4 shows that $\max \notin \mathcal{M}_d(\{0, 1, \ldots, d-1\})$, with a bound on the L_∞ error from this function class. First, however, we present Theorem 3, connecting the average of subpool maxes to order statistics.

Theorem 3. *Let* $S(x; r, d) \triangleq \frac{1}{\binom{d}{r}} \sum_{j=1}^{\binom{d}{r}} \omega_{jr}(x)$ *be the average of all subpool maxes of* $x \in \mathbb{R}^d$ *of order* r*. Let* $x_{(j)}$ *(the subscripts being enclosed in parentheses) denote the* j*th largest element of a vector* x *(*order statistics *notation).*

$$S(x; r, d) = \frac{1}{\binom{d}{r}} \sum_{j=1}^{d-r+1} \binom{d-j}{r-1} x_{(j)}. \tag{4}$$

Sketch of Proof. $x_{(j)}$ is the largest value within a subpool if and only if all indices less than j are excluded from that subpool and j is not excluded. For a subpool of size r the $r-1$ remaining values must be amongst the $d-j$ values $x_{(j+1)}, \ldots, x_{(d)}$. Thus, amongst all subpools of size r, there will be $\binom{d-j}{r-1}$ in which the largest value is $x_{(j)}$. □

To cultivate some intuition, here are some corner cases. $S(x; 1, d)$ is a simple average, but $S(x; d-1, d) = ((d-1)x_{(1)} + x_{(2)})/d$, is mostly the max value, with only the second-largest value contributing – a reasonable approximation to $x_{(1)}$. For example, at the point $x = \begin{pmatrix} 0 & 0 & \ldots & 1 \end{pmatrix}$, the error of an order-1 approximation is $x_{(1)} - S(x; 1, d) = 1 - 1/d$. While for an order $d-1$ approximation it is much less, at $x_{(1)} - S(x; d-1, d) = 1/d$. In essence, the average of subpool maxes of an order $r \in R$ give a summary of the quantiles of the distribution via a particular weighted average, with better fidelity to the max for larger r. The idea is demonstrated further in Figure 1.

Theorem 4 gives our main result. In order to build a unitless measure, we take as the input domain the d-dimensional unit cube.

Theorem 4. *Let* $||f||_\infty$ *denote the the* L_∞ *norm over the unit cube:* $||f||_\infty = \sup_{x\in[0,1]^d} |f(x)|$. *Let* $\text{dist}(R) = \min_{m\in\mathcal{M}_d(R)} ||m - \max||_\infty$.

$$\text{dist}(\{0,1\}) = \frac{1}{2}\frac{d-1}{d} \tag{5}$$
$$\text{dist}(\{d-1\}) = 1/(2d-1) \tag{6}$$
$$\text{dist}(\{0,d-1\}) = 1/(2d) \tag{7}$$
$$\text{dist}(\{0,1,d-1\}) = 1/(2(d+1)) \tag{8}$$
$$\text{dist}(\{0,1,d-2,d-1\}) = 1/d^2 \tag{9}$$
$$\text{dist}(\{0,1,2,\ldots,d-1\}) = 1/2^d. \tag{10}$$

Sketch of Proof. The idea of the proof is as follows. We first establish symmetry as a property of any optimal estimator. Then we assume that the L_∞ norm of the error is characterized by a few key inputs (corners of the unit cube). Under this conjecture, the norm is optimized by evaluating the error at each such point, recognizing the tension between them, and finding the coefficients which equate them. Given the conjectured error, we then prove it is optimal by contradiction using a variant of Farkas' lemma. □

Equation 5 gives a linear model as a baseline for the max. The error is high, and in higher dimensions, not much different to a constant model (obviously, the least error that an intercept alone can obtain is $.5 = \text{dist}(\{0\})$).

Equation 6 shows that error declining with dimensionality is achievable with a higher order term. Contrasting Equation 7 to Equation 6 quantifies the additivity of the intercept. Including an intercept helps, but primarily in low dimensions, which makes sense as an intercept does not scale with dimensionality. Since the intercept requires negligible computation, we assume its inclusion subsequently. Including the average as a feature entails no meaningful further nonlinearity, and reduces the error from $1/(2d)$ to $1/(2(d+1))$ (Equation 8), thus we assume its inclusion subsequently.

Equation 9 is important for understanding how the error falls with the addition of further strongly dimension-sensitive terms: appending $d-2$ to $\{0,1,d-1\}$ improves the rate of convergence from $O(1/d)$ to $O(1/d^2)$. Equation 10 gives the best-case rate of convergence: if *all* lower order terms are included, then the error is $O(1/2^d)$. Contrasting this with Equation 9, we see that the inclusion of many lower order terms is apparently necessary for low error. Qualitatively, Equation 10 implies that $\max \notin \mathcal{M}_d(\{0,1,\ldots,d-1\})$, though it can be approximated well within the function class.

Let $f_R^\star : \mathbb{R}^d \to \mathbb{R}$ denote the optimal estimate based on terms in R. L_∞ error can be high, even if $f_R^\star \approx \max$ on most of the domain. If the the high error could not be realized in practice, because the measure of the domain on which it arises is miniscule, then our result would be only a technicality with little practical relevance. Theorem 5 shows that this is not the case, with a lower bound on the measure of a set on which the L_1 norm is high.

Theorem 5. *For* $\epsilon < \text{dist}(R)/2$, *let* $W(\epsilon;R) = \{x \in [0,1]^d : |x_{(1)} - f_R^\star(x)| \geq \epsilon\}$ *denote the subset of the unit cube where the error of* $f_R^\star$ *is at least* ϵ. *Then for all* R *with* $0 \in R$, $\text{vol}(W(\epsilon;R)) \geq (\text{dist}(R)/2 - \epsilon)^d$, *where* vol *is the Lebesgue measure over* $[0,1]^d$.

In Appendix C we solve for the L_2 error of Equation 10 and find that it is also not zero. Thus: not only can the error be high, it is moreover high on average, and at many points on the domain.

5 Experimental evidence on the relevance of $\mathcal{M}_d(R)$

Each $\mathcal{M}_d(R)$ is a subset of all possible deep neural networks of a given depth. It would harm the analysis of this paper if a more general class of purely feedforward ReLU approximations can achieve significantly less error. This section presents experimental evidence that in approximating the max function, $\mathcal{M}_d(R)$ is an adequate proxy for all networks of the same depth. It does this by showing that additional capacity does not appear to improve the quality of the approximation.

In order to do this, we coin a new function class that generalizes $\mathcal{M}_d(R)$. Every purely feedforward DNN f, can be written as

$$f = (x \mapsto W_k x + b_k) \circ \mathrm{ReLU} \circ \ldots \mathrm{ReLU} \circ (x \mapsto W_0 x + b_0) \tag{11}$$

where $W_j, b_j, j = 0, 1, \ldots, k$ are the weights and biases of the linear layers, and k is the depth. Appendix E presents a concrete and efficient algorithm for representing $f^\star_R$ in the form of Equation 11. For example, via this procedure with $d = 9$ and $R = \{0, 1, 7, 8\}$, $f^\star_R$ can be computed by a purely feedforward ReLU network with hidden layer widths $78, 122, 182$.

For f given in Equation 11, let $w(f) \in \mathbb{N}^{k+1}$ contain the number of columns of $W_0, W_1, \ldots, W_k$, that is, the widths of the hidden layers. Let $\mathcal{G}_k$ denote the set of all depth k purely feedforward DNNs, and for $\mu > 0$ let

$$\mathcal{G}_d(R, \mu) = \{g \in \mathcal{G}_{\lceil \log_2(\max R) \rceil} : w(g) \leq \mu \times w(f^\star_R)\}. \tag{12}$$

In words: $\mathcal{G}_d(R, \mu)$ is the set of all neural networks that are at most μ times as wide as $f^\star_R$, the optimal estimator in $\mathcal{M}_d(R)$. The properties $\mathcal{M}_d(R) \subseteq \mathcal{G}_d(R, 1), \mu_1 < \mu_2 \implies \mathcal{G}_d(R, \mu_1) \subseteq \mathcal{G}_d(R, \mu_2)$ and $\lim_{\mu \uparrow \infty} \mathcal{G}_d(R, \mu) = \mathcal{G}_{\lceil \log_2(\max R) \rceil}$ characterize the sense in which $\mathcal{G}_d(R, \mu)$ represent a parameterized interpolation between $\mathcal{M}_d(R)$ and all networks of a given depth.

5.1 Experimental setup

Independent test and train datasets are generated uniformly on the unit cube with 10000 rows. L_∞ loss is directly optimized by an Adam optimizer (with PyTorch default parameters). Results are similar with MSE loss. We use Kaiming initialization for weight matrices and a small positive constant initialization for the biases. A batch size of 512 is used throughout, and the data set is shuffled over epochs. Training is stopped when the improvement in the test error of is less than .005 over the preceding five epochs, or after 500 epochs. All computations are run on a mix of inexpensive, consumer-grade graphics processing units (GPUs), and all experiments described in this section can be run in a few GPU-days. We repeat all analyses over ten pseudorandom number generator seeds, with both the data being generated differently, and different randomness in the fitting (e.g. the shuffling over minibatches). In addition to the average, we shade the region enclosed by ± 1.96 standard devations, which can be interpreted as a 95% confidence interval.

5.2 Results

Our experimental evidence consists of assessing how expressiveness of fitted elements of $\mathcal{G}_d(R, \mu)$ depends on μ. If expressiveness, quantified as performance on a randomly-generated problem instance, is not substantially increased for progressively greater μ, then a conclusion that holds for $\mathcal{M}_d(R)$ materially also holds for $\mathcal{G}_d$. As our expressiveness measure, we have two different metrics:

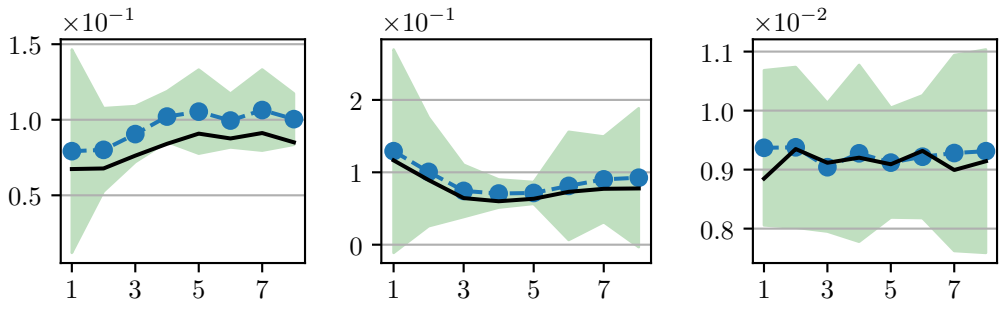
1. $\textsc{err}(R, \mu)$: the empirical L_∞ error of a DNN $\in \mathcal{G}_d(R, \mu)$ optimized to model $x \mapsto x_{(1)}$
2. $\textsc{relerr}(R, \mu)$: the empirical L_∞ error of a DNN $\in \mathcal{G}_d(R, \mu)$ optimized to model $x \mapsto x_{(1)} - f^\star_R(x)$

$\textsc{err}(R, \mu)$ measures directly the approximation error, and finding that it falls reliably with μ for each R would be strong evidence that $\mathcal{M}_d(R)$ is an inadequate function class, since error can be reduced by stepping outside of it. This is not what we see.

$\textsc{relerr}(R, \mu)$ is an indirect error measure that is necessary because although *our* fitting does not find a good optimum does not mean that a superior procedure could not. This measure isolates some of the difficulties that a standard stochastic gradient descent (SGD) procedure might have in modelling max pooling operations.

We focus on μ starting at 1, because this corresponds to the width of $f^\star_R$.[2] Universal approximation theorems (UATs) ensure that that there is a network that achieves arbitrarily low error as $\mu \uparrow \infty$. However, like our

[2]Note however that there are non-width reasons that, even for $\mu = 1$, $\mathcal{M}_d(R) \subsetneq \mathcal{G}_d(R, 1)$. One is that $\mathcal{G}_d(R, \mu)$ allows intercepts in the linear layers, despite not being present in $f^\star_R$ (cf. Appendix E). $\mathcal{G}_d(R, \mu)$ also imposes no low-rank structure



(a) ERR($\{0, 1, d-2, d-1\}, \mu$) (b) ERR($\{d\}, \mu$) (c) RELERR($\{0, 1, d-2, d-1\}, \mu$)

Figure 2: Fitting error and model size in $d = 9$ for several R. d is the problem dimension. ERR and RELERR are described in subsection 5.2 and capture, respectively the empirical error of directly fitting the max function, and the empirical error of fitting the max function minus $f_R^\star$ (the optimal estimator in $\mathcal{M}_d(R)$), given general networks with width proportional to μ. Plotted here are error measures as a function of $\mu \in \{1, 2, \ldots, 8\}$. Test errors are in blue, train in black. Green shading gives a 95% confidence interval for test error over 10 pseudorandom number generator seeds

SGD analysis, UATs make make no guarantees about fitting. Thus, we really have little guidance about how high of a μ could be required. We present μ as large as 8, based on GPU memory considerations.

Figure 2 introduces our main experimental result: fitting error does not reliably fall as μ rises. Figure 2(a) hints at the difficulty of SGD fitting for ERR – with $R = \{0, 1, 7, 8\}$ in $d = 9$ a criterion of 0.01235 ($= 1/d^2$ cf. Equation 9) is achievable at $\mu = 1$, however no μ comes near this value. We further examine this trend in Figure 2(b), where in fact zero error is attainable, however the observed error is little improved. It appears that a standard SGD procedure has trouble even achieving dist(R). Thus, predicting $x_{(1)} - f_R^\star(x)$ is plausibly easier than predicting $x_{(1)}$ alone. This is why in Figure 2(c) we examine RELERR(R, μ). Here as well, we see practically no reduction in error with larger μ.

An estimator that achieves an RELERR(R, μ), in turn implies a realizable ERR$(R, \mu + 1)$, by adding $f_R^\star$ to both sides. Thus, we take Figure 2(c) to be strong evidence that more expressiveness does not practically enable lower error in approximating $x_{(1)}$.

Figure 3 shows the same pattern of incremental width not being effectively utilized to reduce error across model sizes. This figure focuses on RELERR with $d = 10$ at $R = \{0, 1, d-1\}$, $\{0, 1, d-2, d-1\}$, and $\{0, 1, 2, 3, \ldots, d-1\}$.

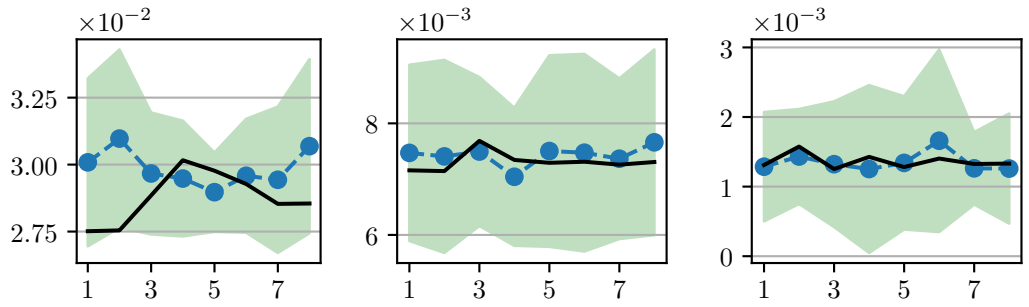
This analysis shows that greater width does not seem to decrease approximation error. From this, we conclude that although our main results are proven only for $\mathcal{M}_d(R)$, empirically they translate well to more general and powerful function classes within the space of all purely feedforward networks.

5.3 The complexity of $\mathcal{M}_d(R)$

Equation 6 and Equation 10 represent quite disparate orders of estimation error. This surprising dispersion is better understood by appreciating just how many different estimators $\mathcal{M}_d(R)$ covers for different R.

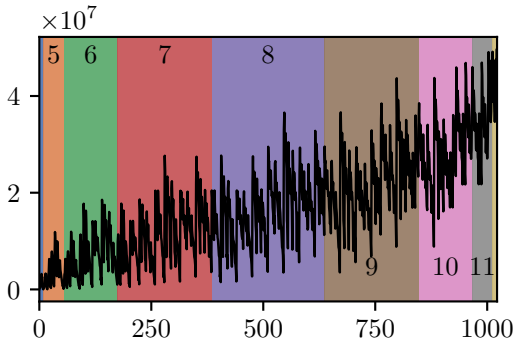
The complexity of $\mathcal{M}_d(\{0, 1, 2, \ldots, d-1\})$ is high, whilst elements of $\mathcal{M}_d(\{0, 1, d-1\})$ can be simple. Figure 4 demonstrates this by plotting the number of parameters in a deep neural network representation of an R-estimator (described in Appendix E).

on the weight matrices, despite a straightforward representation of the theoretical weights as the outer product of matrices roughly one quarter as large (via the quadrupling of layer sizes implied by Equation 2). The larger is the class of models that does not substantially reduce error, the more conservative are our experimental findings.

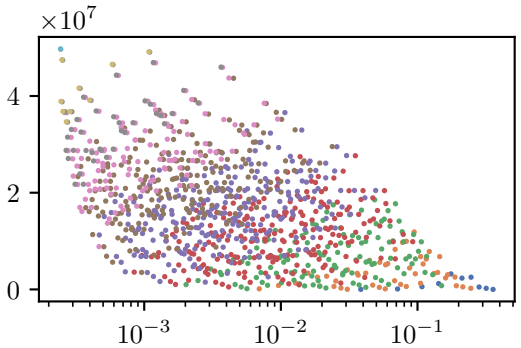


(a) RELERR($\{0, 1, d-1\}, \mu$) (b) RELERR($\{0, 1, d-2, d-1\}, \mu$) (c) RELERR($\{0, 1, 2, 3, \ldots, d-1\}, \mu$)

Figure 3: Fitting error and model size in $d = 10$. RELERR is described in subsection 5.2 and captures the empirical error of fitting the max function minus our estimator $f_R^\star$, given general networks with width proportional to μ. Plotted here are error measures as a function of $\mu \in \{1, 2, \ldots, 8\}$. Test errors are in blue, train in black. Green shading gives a 95% confidence interval for test error over 10 pseudorandom number generator seeds.



(a) Parameter count for R ordered by size, then lexicographically within sizes. Different $|R|$ are denoted with differing background colors.



(b) L_∞ error and parameter count. Marker color is as in Figure 4(a).

Figure 4: Model size and error in $d = 12$

Figure 4(a) relates the parameter counts to R. This ranges from less than 3500 parameters for $R = \{0, 1, 2\}$, to nearly 50 million for $R = \{0, 1, \ldots, d-1\}$. Figure 4(b) further relates the model size to the L_∞ error bound computed in Appendix D to convey a sense of how many parameters are needed to achieve a given error. In both plots, we group models by $|R|$, indicated by color. These figures demonstrate that the complexity of approximations can vary widely in the complexity (as quantified by the number of parameters), and also accuracy, indeed with there being better and worse ways to allocate parameters through a network.

6 Conclusion

Motivated by a marked trend in the design of computer vision architectures, we have posed and answered the question: can max pooling be replaced by linear mappings composed with ReLU activations? And when would doing so give a model that is considerably different?

To do this, we first showed why distance in intermediate feature space is the correct notion of comparison. Next, we established a simple baseline: max pooling with kernel size of d can be computed by a block

of $\log_2(d)$ depth and moderate width. We next introduced subpool max averages as a tractable class of approximators, and proved that the max function in d dimensions cannot be written as the linear combination of subpool max averages of order $< d$, though the error can be made as low as $1/2^d$. By establishing experimentally that the class of subpool max averages was not significanly less expressive than more general function classes, we extended our analysis to wider networks not constrained to have a fixed weight pattern. As a byproduct of this analysis, we are able to also visualize the complexity of all approximators.

In future, we hope to further examine practical implications of this analysis, establishing experimentally that there me be some non-accuracy reasons to prefer max pooling, such as adversarial robustness. Some preliminary evidence on these conjectures is presented in subsection F.1.

References

Raman Arora, Amitabh Basu, Poorya Mianjy, and Anirbit Mukherjee. Understanding deep neural networks with rectified linear units. *International Conference on Learning Representations*, 2018. URL http://arxiv.org/abs/1611.01491.

Peter L. Bartlett, Nick Harvey, Christopher Liaw, and Abbas Mehrabian. Nearly-tight VC-dimension and pseudodimension bounds for piecewise linear neural networks. *Journal of Machine Learning Research*, 20 (63):1–17, 2019. URL http://jmlr.org/papers/v20/17-612.html.

Christian Blatter. Nice expression for minimum of three variables? Mathematics Stack Exchange, 2011. URL https://math.stackexchange.com/q/53820.

Y-Lan Boureau, Jean Ponce, and Yann LeCun. A theoretical analysis of feature pooling in visual recognition. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, pp. 111–118, Madison, WI, USA, 2010. URL https://doi.org/10.5555/3104322.3104338.

Steven Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

Walter B. Carver. Systems of linear inequalities. *Annals of Mathematics*, 23(3):212–220, 1922. URL http://www.jstor.org/stable/1967919.

Ronan Collobert. *Large Scale Machine Learning*. PhD thesis, Université Paris VI, 2004.

Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. BinaryConnect: Training Deep Neural Networks with binary weights during propagations. *arXiv e-prints*, November 2015. URL https://doi.org/10.48550/arXiv.1511.00363.

George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, Dec 1989. URL https://doi.org/10.1007/BF02551274.

William Falcon. Pytorch lightning. *GitHub. Note: https://github.com/PyTorchLightning/pytorch-lightning*, 3, 2019.

Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36:193–202, 1980. URL https://doi.org/10.1007/BF00344251.

Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*, 2015. URL http://arxiv.org/abs/1412.6572.

Philipp Grüning and Erhardt Barth. Bio-inspired Min-Nets Improve the Performance and Robustness of Deep Networks. *SVRHM 2021 Workshop @ NeurIPS*, pp. arXiv:2201.02149, January 2022. URL https://arxiv.org/abs/2201.02149.

Boris Hanin and David Rolnick. Complexity of Linear Regions in Deep Networks. *arXiv e-prints*, art. arXiv:1901.09021, Jan 2019. URL https://arxiv.org/abs/1901.09021.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. URL https://doi.org/10.1109/CVPR.2016.90.

Christoph Hertrich, Amitabh Basu, Marco Di Summa, and Martin Skutella. Towards lower bounds on the depth of relu neural networks. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 3336–3348. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper/2021/file/1b9812b99fe2672af746cefda86be5f9-Paper.pdf.

Elad Hoffer, Itay Hubara, and Daniel Soudry. Train longer, generalize better: closing the generalization gap in large batch training of neural networks. In *Advances in Neural Information Processing Systems*, 2017. URL https://proceedings.neurips.cc/paper/2017/file/a5e0ff62be0b08456fc7f1e88812af3d-Paper.pdf.

Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, 1991. ISSN 0893-6080. URL https://doi.org/10.1016/0893-6080(91)90009-T.

Andrew Howard, Ruoming Pang, Hartwig Adam, Quoc V. Le, Mark Sandler, Bo Chen, Weijun Wang, Liang-Chieh Chen, Mingxing Tan, Grace Chu, Vijay Vasudevan, and Yukun Zhu. Searching for mobilenetv3. In *IEEE/CVF International Conference on Computer Vision*, 2019. URL https://doi.org/10.1109/ICCV.2019.00140.

Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009. URL https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

Alex Krizhevsky. Convolutional deep belief networks on CIFAR-10. 05 2012. URL https://www.cs.toronto.edu/~kriz/conv-cifar10-aug2010.pdf.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, May 2017. ISSN 0001-0782, 1557-7317. URL https://doi.org/10.1145/3065386.

Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng. Rectifier nonlinearities improve neural network acoustic models. *ICML Workshop on Deep Learning for Audio, Speech, and Language Processing*, 2013. URL https://ai.stanford.edu/~amaas/papers/relu_hybrid_icml2013_final.pdf.

Sébastien Marcel and Yann Rodriguez. Torchvision the machine-vision package of torch. In *Proceedings of the 18th ACM International Conference on Multimedia*, MM '10, pp. 1485–1488, New York, NY, USA, 2010. Association for Computing Machinery. URL https://doi.org/10.1145/1873951.1874254.

David C. Page. How to train your resnet 8: Bag of tricks, 2018. URL https://myrtle.ai/learn/how-to-train-your-resnet-8-bag-of-tricks/.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. In *Neural Information Processing Systems 2017 Workshop Autodiff Program*, 2017.

Jonas Rauber, Wieland Brendel, and Matthias Bethge. Foolbox: A python toolbox to benchmark the robustness of machine learning models. In *Reliable Machine Learning in the Wild Workshop, 34th International Conference on Machine Learning*, 2017. URL http://arxiv.org/abs/1707.04131.

Maximiliam Riesenhuber and Tomaso Poggio. Hierarchical models of object recognition in cortex. *Nat. Neurosci.*, 2(11):1019–1025, 1999. URL https://doi.org/10.1038/14819.

Thiago Serra, Christian Tjandraatmadja, and Srikumar Ramalingam. Bounding and counting linear regions of deep neural networks. In *International Conference on Machine Learning*. PMLR, 2018. URL http://proceedings.mlr.press/v80/serra18b.html.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In Yoshua Bengio and Yann LeCun (eds.), *International Conference on Learning Representations*, 2015. URL http://arxiv.org/abs/1409.1556.

Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin A. Riedmiller. Striving for Simplicity: The All Convolutional Net. In *International Conference on Learning Representations, ICLR Workshop Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6806.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2016. URL https://doi.org/10.1109/CVPR.2016.308.

Matus Telgarsky. Benefits of depth in neural networks. In *29th Annual Conference on Learning Theory*. PMLR, 2016. URL https://proceedings.mlr.press/v49/telgarsky16.html.