Why neural networks find simple solutions: the many regularizers of geometric complexity

Benoit Dherin[*]
Google
dherin@google.com

Michael Munn[*]
Google
munn@google.com

Mihaela C. Rosca
DeepMind, London
mihaelacr@deepmind.com

David G.T. Barrett
DeepMind, London
barrettdavid@deepmind.com

Abstract

In many contexts, simpler models are preferable to more complex models and the control of this model complexity is the goal for many methods in machine learning such as regularization, hyperparameter tuning and architecture design. In deep learning, it has been difficult to understand the underlying mechanisms of complexity control, since many traditional measures are not naturally suitable for deep neural networks. Here we develop the notion of geometric complexity, which is a measure of the variability of the model function, computed using a discrete Dirichlet energy. Using a combination of theoretical arguments and empirical results, we show that many common training heuristics such as parameter norm regularization, spectral norm regularization, flatness regularization, implicit gradient regularization, noise regularization and the choice of parameter initialization all act to control geometric complexity, providing a unifying framework in which to characterize the behavior of deep learning models.

1 Introduction

Regularization is an essential ingredient in the deep learning recipe and understanding its impact on the properties of the learned solution is a very active area of research [32, 42, 73, 84]. Regularization can assume a multitude of forms, either added explicitly as a penalty term in a loss function [32] or implicitly through our choice of hyperparameters [6, 76, 79, 81, 83], model architecture [57, 59, 60] or initialization [30, 36, 37, 58, 70, 93, 94]. These forms are generally not designed to be analytically tractable, but in practice, regularization is often invoked in the control of model complexity, putting a pressure on a model to discover simple solutions more so than complex solutions.

To understand regularization in deep learning, we need to precisely define model 'complexity' for deep neural networks. Complexity theory provides many techniques for measuring the complexity of a model, such as a simple parameter count, or a parameter norm measurement [4, 22, 63, 72] but many of these measures can be problematic for neural networks [44, 91]. The recently observed phenomena of 'double-descent' [10, 11, 71] illustrates this clearly: neural networks with high model complexity, as measured by a parameter count, can fit training data closely (sometimes interpolating the data exactly), while simultaneously having low test error [71, 91]. Classically, we expect that interpolation of training data is evidence of overfitting, but yet, neural networks seem to be capable of interpolation while also having low test error. It is often suggested that some form of implicit

[*]equal contribution

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

regularization or explicit regularization is responsible for this, but how should we account for this in theory, and what complexity measure is most appropriate?

In this work, we develop a measure of model complexity, called Geometric Complexity (GC), that has properties that are suitable for the analysis of deep neural networks. We use theoretical and empirical techniques to demonstrate that many different forms of regularization and other training heuristics can act to control geometric complexity through different mechanisms. We argue that the geometric complexity provides a convenient proxy for neural network performance.

Our primary contributions are:

- We develop a computationally tractable notion of complexity (Section 2), which we call *Geometric Complexity* (GC), that has many close relationships with many areas in deep learning and mathematics including harmonic function theory, Lipschitz smoothness (Section 2), and regularization theory (Section 4).
- We provide evidence that common training heuristics keep the geometric complexity low, including: (i) common initialization schemes (Section 3) (ii) the use of overparametrized models with a large number of layers (Fig. 2) (iii) large learning rates, small batch sizes, and implicit gradient regularization (Section 5 and Fig. 4) (iv) explicit parameter norm regularization, spectral norm regularization, flatness regularization, and label noise regularization (Section 4 and Fig. 3)
- We show that the geometric complexity captures the double-descent behaviour observed in the test loss as model parameter count increases (Section 6 and Fig. 5).

The aim of this paper is to introduce geometric complexity, explore its properties and highlight its connections with existing implicit and explicit regularizers. To disentangle the effects studied here from optimisation choices, we use stochastic gradient descent without momentum to train all models. We also study the impact of a given training heuristic on geometric complexity in isolation of other techniques to avoid masking effects. For this reason we do not use data augmentation or learning rate schedules in the main part of the paper. In the Supplementary Material (SM) we redo most experiments using SGD with momentum (Section C.7) and Adam (Section C.8) with very similar conclusion. We also observe the same behavior of the geometric complexity in a setting using learning rate schedule, data augmentation, and explicit regularization in conjunction to improve model performance (Section C.6). The exact details of all experiments in the main paper are listed in SM Section B. All additional experiment results and details can be found in SM Section C.

2 Geometric complexity and Dirichlet energy

Although many different forms of complexity measures have been proposed and investigated (e.g., [21, 44, 72]), it is not altogether clear what properties they should have, especially for deep learning. For instance, a number of them like the Rademacher complexity [50], the VC dimension [87], or the simple model parameter count focus on measuring the entire hypothesis space, rather than a specific function, which can be problematic in deep learning [44, 91]. Other measures like the number of linear pieces for ReLU networks [3, 80] or various versions of the weight matrix norms [72] measure the complexity of the model function independently from the task at hand, which is not desirable [75]. An alternative approach is to learn a complexity measure directly from data [56] or to take the whole training procedure over a dataset into account [71]. Recently, other measures focusing on the model function complexity over a dataset have been proposed in [26] and [63] to help explain the surprising generalization power of deep neural networks. Following that last approach and motivated by frameworks well established in the field of geometric analysis [46], we propose a definition of complexity related to the theory of harmonic functions and minimal surfaces. Our definition has the advantage of being computationally tractable and implicitly regularized by many training heuristics in the case of neural networks. It focuses on measuring the complexity of individual functions rather than that of the whole function space, which makes it different from the Radamacher or VC complexity.

Definition 2.1. *Let $g_\theta : \mathbb{R}^d \to \mathbb{R}^k$ be a neural network parameterized by θ. We can write $g_\theta(x) = a(f_\theta(x))$ where a denotes the last layer activation, and f_θ its logit network. The GC of the network*

over a dataset D is defined to be the discrete Dirichlet energy of its logit network:

$$\langle f_\theta,\, D\rangle_G = \frac{1}{|D|}\sum_{x\in D} \|\nabla_x f_\theta(x)\|_F^2, \tag{1}$$

where $\|\nabla_x f_\theta(x)\|_F$ is the Frobenius norm of the network Jacobian.

Note that this definition is well-defined for any differentiable model, not only a neural network, and incorporates both the model function and the dataset over which the task is determined.

Next, we discuss how GC relates to familiar concepts in deep learning.

Geometric complexity and linear models: Consider a linear transformation $f(x) = Ax + b$ from $\mathbb{R}^d$ to $\mathbb{R}^k$ and a dataset $D = \{x_i\}_{i=1}^N$ where $x_i \in \mathbb{R}^d$. At each point $x \in D$, we have that $\|\nabla_x f(x)\|_F^2 = \|A\|_F^2$, hence the GC for a linear transformation is $\langle f_\theta, D\rangle_G = \|A\|_F^2$. Note, this implies that GC for linear transformations (and more generally, affine maps), is independent of the dataset D and zero for constant functions. Furthermore, note that the GC in this setting coincides precisely with the L2 norm of the model weight matrix. Thus, enforcing an L2 norm penalty is equivalent to regularizing the GC for linear models (see Section 4 for more on that point).

Geometric complexity and ReLU networks: For a ReLU network $g_\theta : \mathbb{R}^d \to \mathbb{R}^k$ as defined in Definition 2.1, the GC over a dataset D has a very intuitive form. Since a ReLU network parameterizes piece-wise linear functions [3], the domain can be broken into a partition of subsets $X_i \subset \mathbb{R}^d$ where f_θ is an affine map $A_i x + b_i$. Now denote by D_i the points in the dataset D that fall in the linear piece defined on X_i. For every point x in X_i, we have that $\|\nabla_x f_\theta(x)\|_F^2 = \|A_i\|_F^2$. Since the D_i's partition the dataset D, we obtain

$$\langle f_\theta, D\rangle_G = \sum_i \left(\frac{n_i}{|D|}\right) \|A_i\|_F^2, \tag{2}$$

where n_i is the number of points in the dataset D falling in X_i. We see from Eqn. (2) that for ReLU networks the GC over the whole dataset coincides exactly with the GC on a batch $B \subset D$, provided that the proportion of points in the batch falling into each of the the linear pieces are preserved. This makes the evaluation of the GC on large enough batches a very good proxy to the overall GC over the dataset, and computationally tractable during training.

Geometric complexity and Lipschitz smoothness: One way to measure the smoothness of a function $f : \mathbb{R}^d \to \mathbb{R}^k$ on a subset $X \subset \mathbb{R}^d$ is by its Lipschitz constant; i.e., the smallest $f_L \geq 0$ such that $\|f(x_1) - f(x_2)\| \leq f_L \|x_1 - x_2\|$, for all $x_1,\, x_2 \in X$. Intuitively, the constant f_L measures the maximal amount of variation allowed by f when the inputs change by a given amount. Using the Lipschitz constant, one can define a complexity measure of a function f as the Lipschitz constant f_L of the function over the input domain $\mathbb{R}^d$. Since $\|\nabla_x f(x)\|_F^2 \leq \min(k,d)\|\nabla_x f(x)\|_{op}^2 \leq \min(k,d) f_L^2$ where $\|\cdot\|_{op}$ is the operator norm, we obtain a general bound on the GC by the Lipschitz complexity:

$$\langle f, D\rangle_G \;=\; \frac{1}{|D|}\sum_{x\in D} \|\nabla_x f(x)\|_F^2 \;\leq\; \min(k,d) f_L^2. \tag{3}$$

While the Lipschitz smoothness of a function provides an upper bound on GC, there are a few fundamental differences between the two quantities. Firstly, GC is data dependent: a model can have low GC while having high Lipschitz constant due to the model not being smooth in parts of the space where there is no training data. Secondly, the GC can be computed exactly given a model and dataset, while for neural networks only loose upper bounds are available to estimate the Lipschitz constant.

Geometric complexity, arc length and harmonic maps: Let us start with a motivating example: Consider a dataset consisting of 10 points lying on a parabola in the plane and a large ReLU deep neural network trained via gradient descent to learn a function $f : \mathbb{R} \to \mathbb{R}$ that fits this dataset (Fig. 1). Throughout training the model function seems to attain minimal arc length for a given level of training error. Recalling the formula for arc length of f, which is the integral of over $[-1, 1]$ of $\sqrt{1 + f'(x)^2}$, and using the Taylor approximation $\sqrt{1+x^2} \approx 1 + \frac{x^2}{2}$, it follows that minimizing the arc length is equivalent to minimizing the classic Dirichlet energy:

$$E(f) = \frac{1}{2}\int_\Omega \|\nabla_x f(x)\|^2 dx. \tag{4}$$

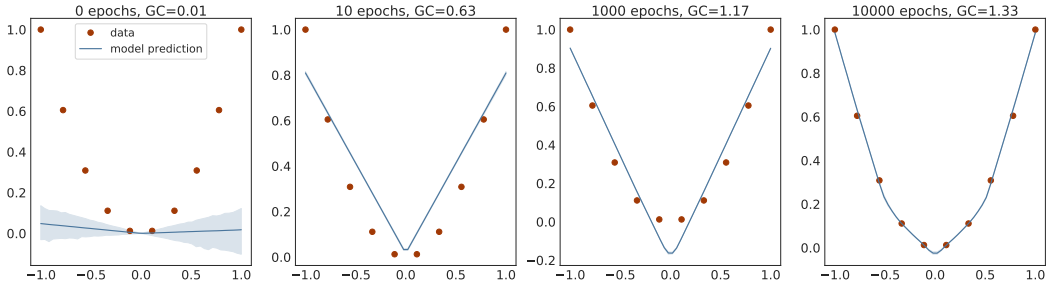

Figure 1: For a large MLP fitting 10 points, the complexity of the function being learned gradually grows in training, while avoiding unnecessary complexity by keeping the function arc length minimal.

where $\Omega = [-1, 1]$. The Dirichlet energy can be thought intuitively of as a measure of the variability of the function f on Ω. Its minimizers, subject to a boundary condition $f_{|\partial\Omega} = h$, are called *harmonic maps*, which are maps causing the "least intrinsic stretching" of the domain Ω [85]. The geometric complexity in Definition 2.1 is an unbiased estimator of a very related quantity

$$\mathbb{E}_X(\|\nabla_x f_\theta(x)\|_F^2) = \int_{\mathbb{R}^d} \|\nabla_x f_\theta(x)\|_F^2 \, p_X(x) dx, \tag{5}$$

where the domain Ω is replaced by the probability distribution of the features p_X and the boundary condition is replaced by the dataset D. We could call the quantity defined in Eqn. (5) the *theoretical geometric complexity* as opposed to the *empirical geometric complexity* in Definition 2.1.

In the remaining sections we provide evidence that common training heuristics do indeed keep the GC low, encouraging neural networks to find intrinsically simple solutions.

3 Impact of initialization on geometric complexity

Parameter initialization choice is an important factor in deep learning. Although we are free to specify exact parameter initialization values, in practice, a small number of default initialization schemes have emerged to work well across a wide range of tasks [30, 37]. Here, we explore the relationship between some of these initialization schemes and GC.

To begin, consider the one dimensional regression example that we introduced in Figure 1. In this experiment, we employed a standard initialization scheme to initialise the parameters: we sample them from a truncated normal distribution with variance inversely proportional to the number of input units and the bias terms were set to zero. We observe that the initialised function on the interval $[-1, 1]$ is very close to the zero function (Fig. 1), and the zero function has zero GC. This observation suggests that initialization schemes that have low initial GC are useful for deep learning.

To explore this further, we consider deep ReLU networks with larger input and output spaces initialized using the same scheme as above, and measure the GC of the resulting model. Specifically, consider the initialized ReLU network given by $f_{\theta_0} : \mathbb{R}^d \to \mathbb{R}^k$ with $d = 150528$ and $k = 1000$, with parameter initialization θ_0, and varying network depth. We measure the ReLU network output size by recording the mean and maximum output values, evaluated using input values along the line $P_1 + (P_2 - P_1)x$ with $x \in [0, 1]$ between two diagonal points P_1 and P_2 of the hyper-cube $[-1, 1]^d$. We observe that these ReLU networks initialize to functions close to the zero function, and become progressively closer to a zero valued function as the number of layer increases (Fig. 2). For ReLU networks, this is not entirely surprising. With biases initialised to zero, we can express a ReLU network in a small neighborhood of a given point $x \in \mathbb{R}^d$ as a product of matrices $f_{\theta_0}(x) = W_1 P_2 W_2 P_3 W_3 \cdots P_l W_l x$, where the W_i's are the weight matrices and the P_i's are diagonal matrices with 0 and 1 on their diagonals. This representation makes it clear that at initialization the ReLU network passes through the origin; i.e., $f_{\theta_0}(0) = 0$. Furthermore, with weight matrices initialised around zero, using a scaling that can reduce the spread of the distribution as the matrices grow, we can expect that deeper ReLU networks generated by multiplying a large number of small-valued weights, can produce output values close to zero (for input values taken from a hyper-cube $[-1, 1]^d$). We extend these results further to include additional initialization schemes and experimentally confirm that the GC can be brought close to zero with a sufficient number of layers. In fact, this is true not

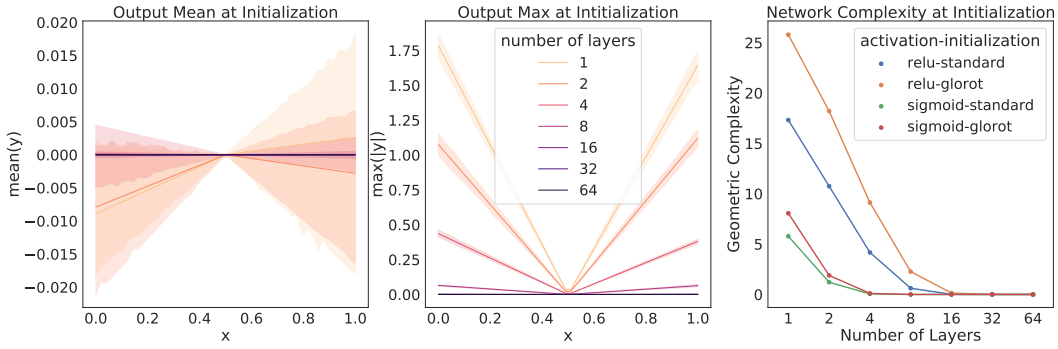

Figure 2: MLP's given by $y = f_{\theta_0}(x)$ initialize closer to the zero function and closer to zero GC, as the number of layers increases. **Left and Middle:** The ReLU MLP's initialized with the standard scheme are evaluated along a range of normalised input values. **Right:** GC is computed on a dataset D of 100 normalized data points. All MLP's have 500 neurons per layer.

only for ReLU networks with the standard initialization scheme, but for a number of other common activation functions and initialization setups [30, 37], and even on domains much larger than the normalized hyper-cube (Fig. 2 and SM Section C.1).

4 Impact of explicit regularization on geometric complexity

Next, we explore the relationship between GC and various forms of explicit regularization using a combination of theoretical and empirical results.

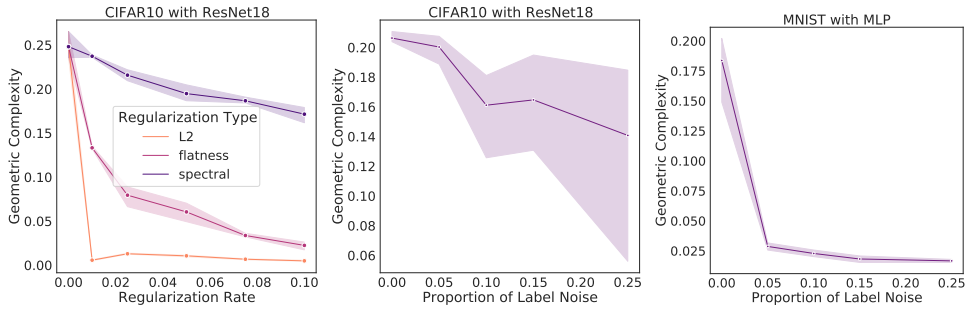

Figure 3: Explicit regularization and GC. **Left:** As the L2, flatness, and spectral regularization increase, GC decreases. (See SM Section C.5 for additional experiments on explicit regularization with ranges targeted to each regularization type.) **Middle and Right:** As label noise regularization increases, GC decreases.

L2 regularization: In L2 regularization, the Euclidean norm of the parameter vector is explicitly added to a loss function, so as to penalize solutions that have excessively large parameter values. This is one of the simplest, and most widely used forms of regularization. For linear models ($f_\theta(x) = Ax + b$), we saw in Section 2 that the GC coincides with the Frobenius norm of the matrix A. This means that standard L2 norm penalties on the weight matrix coincide in this case with a direct explicit regularization of the GC of the linear model. For non-linear deep neural networks, we cannot directly identify the L2 norm penalty with the model GC. However, in the case of ReLU networks, for each input point x, the network output y can be written in a neighborhood of x as $y = P_l W_l \dots P_1 W_1 x + c$, where c is a constant, the P_i's are diagonal matrices with 0 and 1 on the diagonal, and the W_i's are the network weight matrices. This means that the derivative at x of the network coincides with the matrix $P_l W_l \dots P_1 W_1$. Therefore, the GC is just the Frobenius norm of the product of matrices. Now, an L2 penalty $\|W_l\|_F^2 + \dots + \|W_1\|_F^2$ encourages small numbers in the values of the weight matrices, which in turn is likely to encourage small numbers for the values in the product $P_l W_l \dots P_1 W_1$, resulting in a lower GC.

We can demonstrate this relationship empirically, by training a selection of neural networks with L2 regularization, each with a different regularization strength. We measure the GC for each network at the time of maximum test accuracy. We observe empirically that strengthening L2 regularization coincides with a decrease in GC values (Fig. 3).

Lipschitz regularization via spectral norm regularization: A number of explicit regularization schemes have been used to tame the Lipschitz smoothness of the model function and produce smoother models with reduced test error [2, 3, 23, 25, 35, 49, 67, 75, 90]. Smoothness regularization has also been shown to be beneficial outside the supervised learning context, in GANs [15, 67, 92] and reinforcement learning [12, 31]. One successful approach to regularising the Lipschitz constant of a neural network with 1-Lipschitz activation functions (e.g. ReLU, ELU) is to constrain the spectral norm of each layer of the network (i.e., the maximal singular values $\sigma_{\max}(W_i)$ of the weight matrices W_i), since the product of the spectral norms of the networks weight matrices is an upper bound to the Lipschitz constant of the model: $f_L \leq \sigma_{\max}(W_1) \cdots \sigma_{\max}(W_l)$. Using inequality (3), we see that any approach that constrains the Lipschitz constant of the model constrains GC. To confirm this theoretical prediction, we train a ResNet18 model on CIFAR10 [54] and regularize using spectral regularization [90], an approach which adds the regularizer $\frac{\lambda}{2} \sum_l (\sigma_{\max}(W_l))^2$ to the model loss function. We observe that GC decreases as the strength of spectral regularization λ increases (Fig. 3).

Noise regularization: The addition of noise during training is known to be an effective form of regularization. For instance, it has been demonstrated [13] that the addition of noise to training labels during the optimisation of a least-square loss using SGD exerts a regularising pressure on $\sum_{(x,y)\in D} \|\nabla_\theta f_\theta(x)\|^2/|D|$. Here, we demonstrate empirically that the GC of a ResNet18 trained on CIFAR10 reduces as the proportion of label noise increases (Fig. 3, middle). The same is true of an MLP trained on MNIST [19] (Fig. 3, right). In Section 5, we provide a theoretical argument which justifies these experiments, showing that a regularizing pressure on $\|\nabla_\theta f_\theta(x)\|^2$ transfers to a regularizing pressure on $\|\nabla_x f_\theta(x)\|^2$. Thus, label noise in SGD in turn translates into a regularizing pressure on the GC in the case of neural networks.

Flatness regularization: In flatness regularization, an explicit gradient penalty term, $\|\nabla_\theta L_B\|^2$ is added to the loss (where L_B is the loss evaluated across a batch B). It has been observed in practice that flatness regularization can be effective in many deep learning settings, from supervised learning [27, 81] to GAN training [5, 66, 69, 74, 76]. Flatness regularization penalizes learning trajectories that follow steep slopes across the loss surface, thereby encouraging learning to follow shallower slopes toward flatter regions of the loss surface. We demonstrate empirically that GC decreases as the strength of flatness regularization increases (Fig. 3). In the next section, we will provide a theoretical argument that flatness regularization can control GC.

Explicit GC regularization and Jacobian regularization: All the forms of regularization above have known benefits for improving the test accuracy in deep learning. As we saw, they all also implicitly regularize GC. This raises the question as whether regularizing for GC directly and independently of any other mechanism is sufficient to improve model performance. Namely, we can add GC computed on the batch to the loss $L_{\text{reg}}(\theta) = L_B(\theta) + \alpha/B \sum_{x\in B} \|\nabla_x f_\theta(x)\|_F^2$. This is actually a known form of explicit regularization, called **Jacobian regularization**, which has been correlated with increased generalization but also robustness to input shift [42, 84, 89, 90]. [84] specifically shows that adding Jacobian regularization to the loss function can lead to an increase in test set accuracy (their Tables III, IV, and V). In SM Section C.4 we train a MLP on MNIST and a ResNet18 on CIFAR10 regularized explicitly with the geometric complexity. We observe an increase of test accuracy and a decrease of GC with more regularization. Related regularizers include *gradient penalties* of the form $\sum_x (||\nabla_x f_\theta(x)|| - K)^2$ which have been used for GAN training [25, 35, 48]. We leave the full investigation of the importance of GC outside supervised learning for future work.

5 Impact of implicit regularization on geometric complexity

Implicit regularization is a hidden form of regularization that emerges as a bi-product of model training. Unlike explicit regularization, it is not explicitly added to a loss function. In deep learning settings where no explicit regularization is used, it is the only form of regularization. Here, we use a combination of theoretical and empirical results to argue that some recently identified implicit

regularization mechanisms in gradient descent [6, 61, 81] exert a regularization pressure on geometric complexity. Our argument proceeds as follows: 1) we identify a mathematical term (the implicit gradient regularization term) that characterizes implicit regularization in gradient descent, 2) we demonstrate that this term depends on model gradients, 3) we identify the conditions where model gradient terms apply a regularization pressure on geometric complexity.

Step 1: The implicit regularization that we consider emerges as a bi-product of the discrete nature of gradient descent updates. In particular it has been shown that a discrete gradient update $\theta' = \theta - h\nabla_\theta L_B(\theta)$ over a batch of data implicitly minimizes a modified loss, $\widetilde{L}_B = L_B + \frac{h}{4}\|\nabla_\theta L_B\|^2$, where the second term is called the Implicit Gradient Regularizer (IGR) [6]. Gradient descent optimisation is better characterized as a continuous flow along the gradient of the modified loss, rather than the original unmodified loss. By inspection, the IGR term implicitly regularizes training toward trajectories with smaller loss gradients toward flatter region on the loss surface [6] .

Step 2: Next, we develop this implicit regularizer term for a multi-class classification cross-entropy loss term. We can write (see SM Section A.2 for details):

$$\widetilde{L}_B = L_B + \frac{h}{4B}\left(\frac{1}{B}\sum_{x,i}\epsilon_x^i(\theta)^2\|\nabla_\theta f_\theta^i(x)\|^2\right) + \frac{h}{4}A_B(\theta) + \frac{h}{4}C_B(\theta), \tag{6}$$

where $C_B(\theta)$ measures the *batch gradient alignment*

$$C_B(\theta) = \frac{1}{B^2}\sum_{(x,y)\neq(x',y')}\langle\nabla_\theta L(x,y,\theta), \nabla_\theta L(x',y',\theta)\rangle\,. \tag{7}$$

and $A_B(\theta)$ measures the *label gradient alignment*:

$$A_B(\theta) = \frac{1}{B^2}\sum_{x\in B}\sum_{i\neq j}\langle\epsilon_x^i\nabla_\theta f_\theta^i(x), \epsilon_x^j\nabla_\theta f_\theta^j(x)\rangle. \tag{8}$$

where $\epsilon_x^i(x) = a(f_\theta(x))^i - y^i$ is the signed residual and a denotes the activation function of the last layer. Note that this residual term arises in our calculation using $\nabla_\theta L(x,y,\theta) = \epsilon_x^1(\theta)\nabla_\theta f_\theta^1(x) + \cdots + \epsilon_x^k(\theta)\nabla_\theta f_\theta^k(x)$ (see SM Section A.1, Eqn. (14)). This development also extends to other widely used loss functions such as the cross entropy and least-square loss.

Step 3: Now, from the modified loss in Eqn. (6), we can observe the conditions under which SGD puts an implicit pressure on the gradient norms $\|\nabla_\theta f_\theta^i(x)\|^2$ to be small: the batch gradient alignment and label gradient alignment terms must be small relative to the gradient norms, or positive valued.

We also derive conditions under which this implicit pressure on $\|\nabla_\theta f_\theta(x)\|_F^2$ transfers to a regularizing pressure on $\|\nabla_x f_\theta(x)\|_F^2$ (see SM Section A.3 for the proof):.

Theorem 5.1. *Consider a logit network $f_\theta : \mathbb{R}^d \to \mathbb{R}^k$ with l layers, then we have the following inequality*

$$\|\nabla_x f_\theta(x)\|_F^2 \leq \frac{\|\nabla_\theta f_\theta(x)\|_F^2}{T_1^2(x) + \cdots + T_l^2(x)}, \tag{9}$$

where T_i is the transfer function for layer i given by

$$T_i(x,\theta) = \frac{1}{\sqrt{\min(d,k)}}\frac{\sqrt{1+\|h_i(x)\|_2^2}}{\sigma_{\max}(W_i)\sigma_{\max}(h_i'(x))}, \tag{10}$$

where h_i is the subnetwork to layer i and $\sigma_{\max}(A)$ is the maximal singular value of matrix A (i.e., its spectral norm).

Here, we can see that for any settings where the sum $T_1^2(x) + \cdots + T_l^2(x)$ of squared transfer functions diminishes slower than the gradient norm $\|\nabla_\theta f_\theta(x)\|_F^2$ during training, we expect that implicit gradient regularization will apply a regularization pressure on GC.

An immediate prediction arising under these conditions is that the size of the regularization pressure on geometric complexity will depend on the implicit regularization rate h/B in Eqn. (6) (Note that the

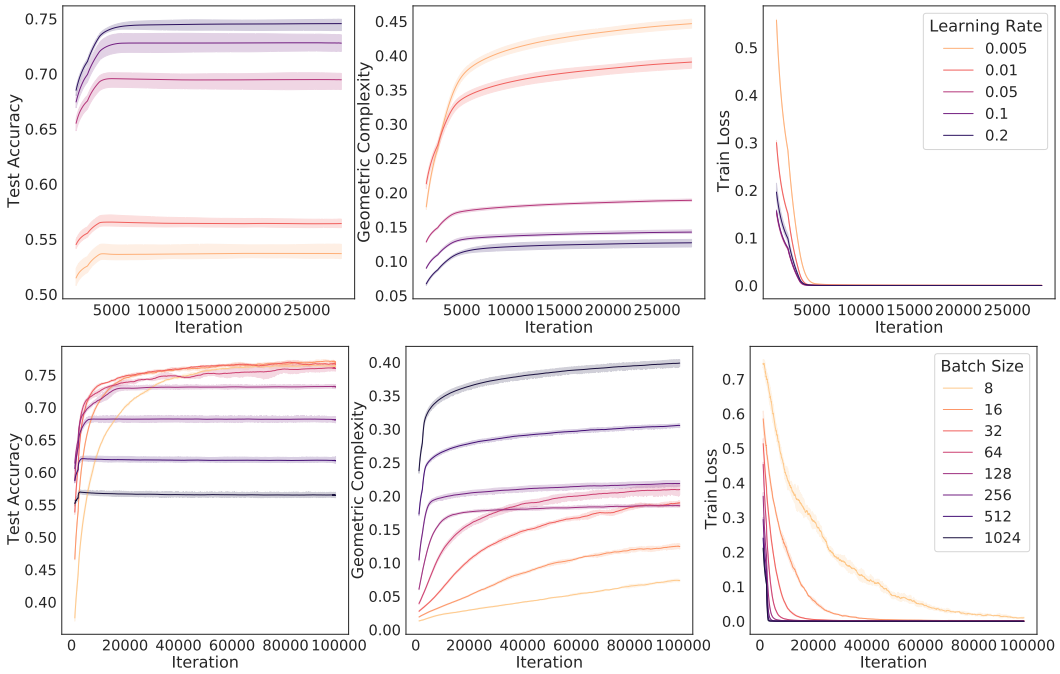

Figure 4: Impact of IGR when training ResNet18 on CIFAR10. **Top row:** As IGR increases through higher learning rates, GC decreases. **Bottom row:** Similarly, lower batch size leads to decreased GC.

ratio h/B has been linked to implicit regularization strength in many instances [33, 82, 65, 55, 61]). Specifically, under these conditions, networks trained with larger learning rates or smaller batch sizes, or both, will apply a stronger regularization pressure on geometric complexity,

We test this prediction by performing experiments on ResNet18 trained on CIFAR10 and show results in Figure 4. The results show that while all models achieve a zero train loss, consistently the higher the learning rate, the higher the test accuracy and the lower the GC. Similarly, we observe that the lower the batch size, the lower the GC and the higher the test accuracy.

The increased performance of lower batch sizes has been long studied in deep learning, under the name 'the generalisation gap' [41, 47]. Crucially however, this gap was recently bridged [28], showing that full batch training can achieve the same test set performance as mini-batch gradient descent. To obtain these results, they use an explicit regularizer similar to the implicit regularizer in Eqn. (6), introduced to compensate for the diminished implicit regularization in the full batch case. Their results further strengthen our hypothesis that implicit regularization via GC results in improved generalisation.

6 Geometric complexity and double descent

When complexity is measured using a simple parameter count, a double descent phenomena has been consistently observed: as the number of parameters increases, the test accuracy decreases at first, before increasing again, followed by a second descent toward a low test error value [10, 11, 71]. An excellent overview of the double descent phenomena in deep learning can be found in [10], together with connections to smoothness as an inductive bias of deep networks and the role of optimisation.

To explore the double descent phenomena using GC we follow the set up introduced in [71]: we train multiple ResNet18 networks on CIFAR10 with increasing layer width, and show results in Fig. 5. We make two observations: first, like the test loss, GC follows a double descent curve as width increases; second, when plotting GC against the test loss, we observe a U-shape curve, recovering the traditional expected behaviour of complexity measures. Importantly, we observe the connection between the generalisation properties of overparametrized models and GC: after the critical region, increase in model size leads to a decrease in GC. This provides further evidence that GC is able to capture model

capacity in a meaningful way, suggestive of a reconciliation of traditional complexity theory and deep learning.

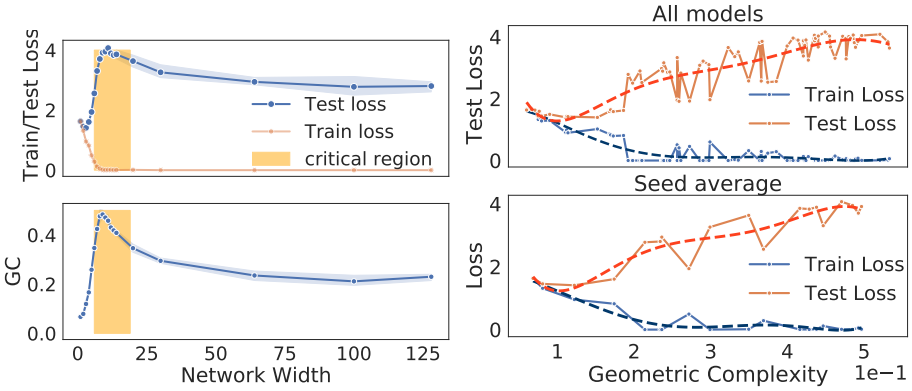

Figure 5: Double descent and GC. **Left:** GC captures the double descent phenomenon. **Right:** GC captures the traditional U-shape curve, albeit with some noise, showing (top) GC vs Test Loss for all models and (bottom) GC vs Test Loss averaged across different seeds. We fit a 6 degree polynomial to the curves to showcase the trend.

7 Related and Future Work

The aim of this work is to introduce a measure which captures the complexity of neural networks. There are many other approaches aiming at doing so, ranging from naive parameter count and more data driven approaches [43, 56, 63] to more traditional measures such as the VC dimension and Rademacher complexity [9, 44, 53, 86] which focus on the entire hypothesis space.

As we saw in Fig. 1, the interpolating ReLU network with minimal GC is also the piecewise linear function with minimal volume or length [20]. In 1D this minimal function can be described only with the information given by the data points. This description with minimal information is reminiscent of the Kolmogorov complexity [78] and the minimum description length [39], and we believe that the exact relationship between these notions, GC, and minimal volume is worth investigating. Similarly, [1] argues that flat solutions have low information content, and for neural networks these flat regions are also the regions of low loss gradient and thus of low GC, as explained by the Transfer Theorem in Section 5. Another recent measure of complexity is the *effective dimension* which is defined in relation to the training data, but is computed using the spectral decomposition of the loss Hessian [63], making flat regions in the loss surface also regions of low complexity w.r.t. this measure. This hints toward the effective dimension being related to GC. Note that similarly to GC, the effective dimension can also capture the double descent phenomena. The effective dimension is an efficient mechanism for model selection [63], which our experiments seem also to indicate may be the case for GC. Note that the generalized degrees of freedom (which considers the sensitivity of a classifier to the labels, rather than to the features as GC does) explored in [34] in the context of deep learning also captures the double-descent phenomena.

GC is close to considerations about smoothness [75] and the Sobolev norm implicit regularization [61]. While GC and Lipchitz smoothness are connected, here we focused on a tractable quantity for neural networks and its connections with existing regularizers. Smoothness regularization has been particularly successful in the GAN literature [16, 35, 48, 67, 92], and the connection between Lipschitz smoothness and GC begs the question of whether their success is due to their implicit regularization of GC, but we leave the application of GC outside supervised learning for future work.

The Dirichlet energy in Eqn. (4) is a well-known quantity in harmonic function theory [24] and a fundamental concept throughout mathematics, physics and, more recently, 3D modeling [85], manifold learning [17], and image processing [29, 77]. Minimizers of the Dirichlet energy are harmonic and thus enjoy certain guarantees in regularity; i.e., any such solution is a smooth function. It has been demonstrated that for mean squared error regression on wide-shallow ReLU networks, gradient descent is biased towards smooth functions at interpolation [45]. Similarly, our work

suggests that neural networks, through a mechanism of implicit regularization of GC, are biased towards minimal Dirichlet energy and thus encourage smooth interpolation. It may be interesting to understand how the relationship between GC, Dirichlet energy, and harmonic theory can help futher improve the learning process, in particular in transfer learning, or OOD generalization.

8 Discussion

In terms of *limitations*, the theoretical arguments presented in this work have a focus on ReLU activations and DNN architectures, with log-likelihood losses coming from the exponential family, such as multi-dimensional regression with least-square losses, or multi-class classification with cross-entropy loss. The experimental results are obtained on the image datasets MNIST and CIFAR using DNN and ResNet architectures.

In terms of *societal impact*, we are not introducing new training methods, but focus on providing a better understanding of the impact of common existing training methods. We hope that this understanding will ultimately lead to more efficient training techniques. While existing training methods may have their own risk, we do not foresee any potential negative societal impact of this work.

In conclusion, altogether, geometric complexity provides a useful lens for understanding deep learning and sheds light into why neural networks are able to achieve low test error with highly expressive models. We hope this work will encourage further research around this new connection, and help to better understand current best practices in model training as well as discover new ones.

Acknowledgments and Disclosure of Funding

We would like to thank Chongli Qin, Samuel Smith, Soham De, Yan Wu, and the reviewers for helpful discussions and feedback as well as Patrick Cole, Xavi Gonzalvo, and Shakir Mohamed for their support.

References

[1] Alessandro Achille and Stefano Soatto. Emergence of invariance and disentanglement in deep representations. In *2018 Information Theory and Applications Workshop (ITA)*, pages 1–9, 2018.

[2] Michael Arbel, Danica J Sutherland, Mikołaj Bińkowski, and Arthur Gretton. On gradient regularizers for mmd gans. *Advances in neural information processing systems*, 31, 2018.

[3] Raman Arora, Amitabh Basu, Poorya Mianjy, and Anirbit Mukherjee. Understanding deep neural networks with rectified linear units. In *International Conference on Learning Representations*, 2018.

[4] Sanjeev Arora, Rong Ge, Behnam Neyshabur, and Yi Zhang. Stronger generalization bounds for deep nets via a compression approach. In *International Conference on Machine Learning*, pages 254–263. PMLR, 2018.

[5] David Balduzzi, Sebastien Racaniere, James Martens, Jakob Foerster, Karl Tuyls, and Thore Graepel. The mechanics of n-player differentiable games. In *International Conference on Machine Learning*, 2018.

[6] David G.T. Barrett and Benoit Dherin. Implicit gradient regularization. In *International Conference on Learning Representations*, 2021.

[7] Peter L Bartlett, Stéphane Boucheron, and Gábor Lugosi. Model selection and error estimation. *Machine Learning*, 48(1):85–113, 2002.

[8] Peter L Bartlett, Nick Harvey, Christopher Liaw, and Abbas Mehrabian. Nearly-tight vc-dimension and pseudodimension bounds for piecewise linear neural networks. *The Journal of Machine Learning Research*, 20(1):2285–2301, 2019.

[9] Peter L Bartlett and Shahar Mendelson. Rademacher and gaussian complexities: Risk bounds and structural results. *Journal of Machine Learning Research*, 3(Nov):463–482, 2002.

[10] Mikhail Belkin. Fit without fear: remarkable mathematical phenomena of deep learning through the prism of interpolation. 2021.

[11] Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116(32), 2019.

[12] Johan Bjorck, Carla P Gomes, and Kilian Q Weinberger. Towards deeper deep reinforcement learning. *arXiv preprint arXiv:2106.01151*, 2021.

[13] Guy Blanc, Neha Gupta, Gregory Valiant, and Paul Valiant. Implicit regularization for deep neural networks driven by an ornstein-uhlenbeck like process. In *Conference on Learning Theory*, 2020.

[14] Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K Warmuth. Learnability and the vapnik-chervonenkis dimension. *Journal of the ACM (JACM)*, 36(4):929–965, 1989.

[15] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis. *arXiv preprint arXiv:1809.11096*, 2018.

[16] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis. In *International Conference on Learning Representations*, 2018.

[17] Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

[18] AIA Chervonenkis and VN Vapnik. Theory of uniform convergence of frequencies of events to their probabilities and problems of search for an optimal solution from empirical data(average risk minimization based on empirical data, showing relationship of problem to uniform convergence of averages toward expectation value). *Automation and Remote Control*, 32:207–217, 1971.

[19] Li Deng. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE signal processing magazine*, 29(6):141–142, 2012.

[20] Benoit Dherin, Michael Munn, and David G.T. Barrett. The geometric occam's razor implicit in deep learning. In *Proceedings on "Optimization for Machine Learning" at NeurIPS Workshops*, 2021.

[21] Gintare Karolina Dziugaite, Alexandre Drouin, Brady Neal, Nitarshan Rajkumar, Ethan Caballero, Linbo Wang, Ioannis Mitliagkas, and Daniel M Roy. In search of robust measures of generalization. *Advances in Neural Information Processing Systems*, 33:11723–11733, 2020.

[22] Gintare Karolina Dziugaite and Daniel M Roy. Computing nonvacuous generalization bounds for deep (stochastic) neural networks with many more parameters than training data. *arXiv preprint arXiv:1703.11008*, 2017.

[23] Gamaleldin Elsayed, Dilip Krishnan, Hossein Mobahi, Kevin Regan, and Samy Bengio. Large margin deep networks for classification. *Advances in neural information processing systems*, 31, 2018.

[24] Lawrence C Evans. *Partial differential equations*, volume 19. American Mathematical Soc., 2010.

[25] William Fedus, Mihaela Rosca, Balaji Lakshminarayanan, Andrew M Dai, Shakir Mohamed, and Ian Goodfellow. Many paths to equilibrium: Gans do not need to decrease a divergence at every step. *arXiv preprint arXiv:1710.08446*, 2017.

[26] Tianxiang Gao and Vladimir Jojic. Degrees of freedom in deep neural networks. In *Proceedings of the Thirty-Second Conference on Uncertainty in Artificial Intelligence*, 2016.

[27] Jonas Geiping, Micah Goldblum, Phil Pope, Michael Moeller, and Tom Goldstein. Stochastic training is not necessary for generalization. In *International Conference on Learning Representations*, 2022.

[28] Jonas Geiping, Micah Goldblum, Phillip E Pope, Michael Moeller, and Tom Goldstein. Stochastic training is not necessary for generalization. *arXiv preprint arXiv:2109.14119*, 2021.

[29] Pascal Getreuer. Rudin-osher-fatemi total variation denoising using split bregman. *Image Processing On Line*, 2:74–95, 2012.

[30] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, 2010.

[31] Florin Gogianu, Tudor Berariu, Mihaela C Rosca, Claudia Clopath, Lucian Busoniu, and Razvan Pascanu. Spectral normalisation for deep reinforcement learning: an optimisation perspective. In *International Conference on Machine Learning*, pages 3734–3744. PMLR, 2021.

[32] Ian J. Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, Cambridge, MA, USA, 2016.

[33] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

[34] Erin Grant and Yan Wu. Predicting generalization with degrees of freedom in neural networks. In *ICML 2022 2nd AI for Science Workshop*, 2022.

[35] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. *Advances in neural information processing systems*, 30, 2017.

[36] Suriya Gunasekar, Jason Lee, Daniel Soudry, and Nathan Srebro. Characterizing implicit bias in terms of optimization geometry. In *International Conference on Machine Learning*, pages 1832–1841. PMLR, 2018.

[37] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015.

[38] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer, 2016.

[39] Geoffrey E Hinton and Richard Zemel. Autoencoders, minimum description length and helmholtz free energy. In J. Cowan, G. Tesauro, and J. Alspector, editors, *Advances in Neural Information Processing Systems*, volume 6. Morgan-Kaufmann, 1994.

[40] Sepp Hochreiter and Jürgen Schmidhuber. Flat minima. *Neural Computation*, 9(1):1–42, 1997.

[41] Elad Hoffer, Itay Hubara, and Daniel Soudry. Train longer, generalize better: closing the generalization gap in large batch training of neural networks. *Advances in neural information processing systems*, 30, 2017.

[42] Judy Hoffman, Daniel A. Roberts, and Sho Yaida. Robust learning with jacobian regularization. 2020.

[43] Yiding Jiang, Dilip Krishnan, Hossein Mobahi, and Samy Bengio. Predicting the generalization gap in deep networks with margin distributions. *arXiv preprint arXiv:1810.00113*, 2018.

[44] Yiding Jiang, Behnam Neyshabur, Hossein Mobahi, Dilip Krishnan, and Samy Bengio. Fantastic generalization measures and where to find them. In *International Conference on Learning Representations*, 2020.

[45] Hui Jin and Guido Montúfar. Implicit bias of gradient descent for mean squared error regression with wide neural networks. *arXiv preprint arXiv:2006.07356*, 2020.

[46] Jürgen Jost and Jèurgen Jost. *Riemannian geometry and geometric analysis*, volume 42005. Springer, 2008.

[47] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016.

[48] Naveen Kodali, Jacob Abernethy, James Hays, and Zsolt Kira. On convergence and stability of gans. *arXiv preprint arXiv:1705.07215*, 2017.

[49] Naveen Kodali, James Hays, Jacob Abernethy, and Zsolt Kira. On convergence and stability of GANs. In *Advances in Neural Information Processing Systems*, 2018.

[50] V. Koltchinskii and D. Panchenko. Rademacher processes and bounding the risk of function learning. In *High Dimensional Probability II*, pages 443–459. Birkhäuser, 1999.

[51] Vladimir Koltchinskii. Rademacher penalties and structural risk minimization. *IEEE Transactions on Information Theory*, 47(5):1902–1914, 2001.

[52] Vladimir Koltchinskii and Dmitriy Panchenko. Rademacher processes and bounding the risk of function learning. In *High dimensional probability II*, pages 443–457. Springer, 2000.

[53] Vladimir Koltchinskii and Dmitry Panchenko. Empirical margin distributions and bounding the generalization error of combined classifiers. *The Annals of Statistics*, 30(1):1–50, 2002.

[54] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[55] Yoonho Lee, Juho Lee, Sung Ju Hwang, Eunho Yang, and Seungjin Choi. In *Advances in Neural Information Processing Systems*, 2020.

[56] Yoonho Lee, Juho Lee, Sung Ju Hwang, Eunho Yang, and Seungjin Choi. Neural complexity measures. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 9713–9724. Curran Associates, Inc., 2020.

[57] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

[58] Yuanzhi Li and Yingyu Liang. Learning overparameterized neural networks via stochastic gradient descent on structured data. *Advances in Neural Information Processing Systems*, 31, 2018.

[59] Henry W Lin, Max Tegmark, and David Rolnick. Why does deep and cheap learning work so well? *Journal of Statistical Physics*, 168(6):1223–1247, 2017.

[60] Chao Ma, Lei Wu, et al. The quenching-activation behavior of the gradient descent dynamics for two-layer neural network models. *arXiv preprint arXiv:2006.14450*, 2020.

[61] Chao Ma and Lexing Ying. The sobolev regularization effect of stochastic gradient descent. 2021.

[62] David MacKay. Bayesian model comparison and backprop nets. *Advances in neural information processing systems*, 4, 1991.

[63] Wesley J Maddox, Gregory Benton, and Andrew Gordon Wilson. Rethinking parameter counting in deep models: Effective dimensionality revisited. *arXiv preprint arXiv:2003.02139*, 2020.

[64] David A McAllester. Pac-bayesian model averaging. In *Proceedings of the twelfth annual conference on Computational learning theory*, pages 164–170, 1999.

[65] Sam McCandlish, Jared Kaplan, Dario Amodei, and OD Team. An empirical model of large-batch training. arxiv 2018. *arXiv preprint arXiv:1812.06162*, 2018.

[66] Lars Mescheder, Sebastian Nowozin, and Andreas Geiger. The numerics of gans. In *Advances in Neural Information Processing Systems*, 2017.

[67] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. In *International Conference on Learning Representations*, 2018.

[68] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of machine learning*. MIT press, 2018.

[69] Vaishnavh Nagarajan and J Zico Kolter. Gradient descent gan optimization is locally stable. In *Advances in neural information processing systems*, 2017.

[70] Vaishnavh Nagarajan and J Zico Kolter. Generalization in deep networks: The role of distance from initialization. *arXiv preprint arXiv:1901.01672*, 2019.

[71] Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. Deep double descent: Where bigger models and more data hurt. *Journal of Statistical Mechanics: Theory and Experiment*, 2021(12):124003, 2021.

[72] Behnam Neyshabur. Implicit regularization in deep learning. *arXiv preprint arXiv:1709.01953*, 2017.

[73] Chongli Qin, James Martens, Sven Gowal, Dilip Krishnan, Krishnamurthy Dvijotham, Alhussein Fawzi, Soham De, Robert Stanforth, and Pushmeet Kohli. Adversarial robustness through local linearization. In *Advances in Neural Information Processing Systems*, 2019.

[74] Chongli Qin, Yan Wu, Jost Tobias Springenberg, Andy Brock, Jeff Donahue, Timothy Lillicrap, and Pushmeet Kohli. Training generative adversarial networks by solving ordinary differential equations. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

[75] Mihaela Rosca, Theophane Weber, Arthur Gretton, and Shakir Mohamed. A case for new neural network smoothness constraints. In *Proceedings on "I Can't Believe It's Not Better!" at NeurIPS Workshops*, volume 137 of *Proceedings of Machine Learning Research*, 2020.

[76] Mihaela Rosca, Yan Wu, Benoit Dherin, and David G.T. Barrett. Discretization drift in two-player games. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139, 2021.

[77] Leonid I Rudin, Stanley Osher, and Emad Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D: nonlinear phenomena*, 60(1-4):259–268, 1992.

[78] J. Schmidhuber. Discovering neural nets with low kolmogorov complexity and high generalization capability. *Neural networks : the official journal of the International Neural Network Society*, 10 5, 1997.

[79] Sihyeon Seong, Yegang Lee, Youngwook Kee, Dongyoon Han, and Junmo Kim. Towards flatter loss surface via nonmonotonic learning rate scheduling. In *Proceedings of the Thirty-Fourth Conference on Uncertainty in Artificial Intelligence*, 2018.

[80] Thiago Serra, Christian Tjandraatmadja, and Srikumar Ramalingam. Bounding and counting linear regions of deep neural networks, 2018.

[81] Samuel L Smith, Benoit Dherin, David G.T. Barrett, and Soham De. On the origin of implicit regularization in stochastic gradient descent. In *International Conference on Learning Representations*, 2021.

[82] Samuel L. Smith, Pieter-Jan Kindermans, and Quoc V. Le. Don't decay the learning rate, increase the batch size. In *International Conference on Learning Representations*, 2018.

[83] Samuel L Smith and Quoc V. Le. A bayesian perspective on generalization and stochastic gradient descent. 2018.

[84] Jure Sokolic, Raja Giryes, Guillermo Sapiro, and Miguel R. D. Rodrigues. Robust large margin deep neural networks. *IEEE Transactions on Signal Processing*, 65:4265–4280, 2017.

[85] Justin Solomon, Leonidas Guibas, and Adrian Butscher. Dirichlet energy for analysis and synthesis of soft maps. In *Computer Graphics Forum*, volume 32, pages 197–206. Wiley Online Library, 2013.

[86] Eduardo D Sontag et al. Vc dimension of neural networks. *NATO ASI Series F Computer and Systems Sciences*, 168:69–96, 1998.

[87] V. N. Vapnik and A. Y. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, XVI(2):264–280, 1971.

[88] VN Vapnik and A Ya Chervonenkis. The method of ordered risk minimization, i. *Avtomatika i Telemekhanika*, 8:21–30, 1974.

[89] Dániel Varga, Adrián Csiszárik, and Zsolt Zombori. Gradient regularization improves accuracy of discriminative models. 2018.

[90] Yuichi Yoshida and Takeru Miyato. Spectral norm regularization for improving the generalizability of deep learning. *ArXiv*, abs/1705.10941, 2017.

[91] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. In *International Conference on Learning Representations*, 2017.

[92] Han Zhang, Ian Goodfellow, Dimitris Metaxas, and Augustus Odena. Self-attention generative adversarial networks. In *International conference on machine learning*, pages 7354–7363. PMLR, 2019.

[93] Yaoyu Zhang, Zhi-Qin John Xu, Tao Luo, and Zheng Ma. A type of generalization error induced by initialization in deep neural networks. In *Mathematical and Scientific Machine Learning*, pages 144–164. PMLR, 2020.

[94] Difan Zou, Yuan Cao, Dongruo Zhou, and Quanquan Gu. Gradient descent optimizes over-parameterized deep relu networks. *Machine Learning*, 109(3):467–492, 2020.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [Yes]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [No] The data is widely available (MNIST and CIFAR10). The code we use to train the model is standard training code using standard neural network architectures (MLP and ResNet18) available out-of-the-box in most ml libraries. We included detailed instructions with all the hyper-parameter used in the supplemental material to reproduce them.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]

(d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] We use the CIFAR10 and MNIST datasets for some of our experimental results. We cite both in the paper, for CIFAR following the guidance on Krizhevsky's homepage: https://www.cs.toronto.edu/ kriz/cifar.html.
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

Supplementary Material

A Proofs for Section 5 **17**

A.1 Gradient structure in the exponential family 17

A.2 Computing the expanded modified loss from Section 5 19

A.3 The Transfer Theorem 20

B Experiment details **23**

B.1 Figure 1: Motivating 1D example 23

B.2 Figure 2: Geometric complexity and initialization 23

B.3 Figure 3: Geometric complexity and explicit regularization 24

B.4 Figure 4: Geometric complexity and implicit regularization 25

B.5 Figure 5: Geometric complexity and double-descent 25

C Additional Experiments **27**

C.1 Geometric complexity at initialization decreases with added layers on large domains 27

C.2 Geometric complexity decreases with implicit regularization for MNIST 28

C.3 Geometric complexity decreases with explicit regularization for MNIST 29

C.4 Explicit geometric complexity regularization for MNIST and CIFAR10 30

C.5 Separate L2, flatness, and spectral regularization experiments for CIFAR10 31

C.6 Geometric complexity in the presence of multiple tuning mechanisms 32

C.7 Geometric complexity in the presence of momentum 34

C.8 Geometric complexity in the presence of Adam 36

D Comparison of the Geometric Complexity to other complexity measures **37**

D.1 Rademacher Complexity 37

D.2 VC dimension 38

D.3 Sharpness and Hessian related measures 39

A Proofs for Section 5

In Section A.1 we derive that for a large class of models comprising regression models with the least-square loss and classification models with the cross-entropy loss, the norm square of the loss gradients has a very particular form (Eqn. 17) that we need to develop the modified loss in Eqn. 6 in Step 2 of Section 5. The detail of this development is given in Section A.2. In Section A.3, we give the proof of Thm. 5.1 also needed in Step 2 of Section 5, which bounds the Frobenius norm of the network Jacobian w.r.t. the input with the network Jacobian w.r.t. the parameters.

A.1 Gradient structure in the exponential family

In order to be able to treat both regression and classification on the same footing, we need to introduce the notion of a transformed target $y = \phi(z)$. In regression ϕ is typically the identity, while for classification ϕ maps the labels $z \in \{1, \ldots, k\}$ onto their one-hot-encoded version $\phi(z) \in \mathbb{R}^k$. We will also need to distinguish between the *logit neural network* $f_\theta : \mathbb{R}^d \to \mathbb{R}^k$ from the *response function* $g_\theta(x) = a(f_\theta(x))$ that models the transformed target. Typically, a is the activation function of the last layer. For instance, for regression both the neural network and its response function

coincide, a being the identity, while for classification $f_\theta(x)$ are the logits and a is typically a sigmoid or a softmax function. Now both regression and classification losses at a data point (x, y) can be obtained as the negative log-likelihood of a conditional probability model

$$L(x, y, \theta) = -\log P(y|x, \theta), \tag{11}$$

The conditional probability model for both regression and classification has the same structure. It is obtained by using the neural network to estimate the natural parameter vector $\eta = f_\theta(x)$ of an exponential family distribution:

$$P(y|x, \theta) = h(y) \exp\left(\langle y, f_\theta(x)\rangle - S(f_\theta(x))\right), \tag{12}$$

where $S(\eta)$ is the log-partition function. For models in the exponential family, the distribution mean coincides with the gradient of the log-partition function: $E(Y|\eta) = \nabla_\eta S(\eta)$. The response function (which is the mean of the conditional distribution) is of the form $g_\theta(x) = \nabla_\eta S(f_\theta(x))$, and the last layer is thus given by $a(\eta) = \nabla_\eta S(\eta)$. For these models, the log-likelihood loss at a data point has the simple form

$$L(x, y, \theta) = S(f_\theta(x)) - \langle y, f_\theta(x)\rangle + \text{constant}. \tag{13}$$

We then obtain immediately that the loss derivative w.r.t. to the parameters and w.r.t. to the input can be written as a sum of the the corresponding network derivatives weighted by the signed residuals:

$$\nabla_\theta L(x, y, \theta) = \epsilon_x^1(\theta)\nabla_\theta f_\theta^1(x) + \cdots + \epsilon_x^k(\theta)\nabla_\theta f_\theta^k(x), \tag{14}$$

$$\nabla_x L(x, y, \theta) = \epsilon_x^1(\theta)\nabla_x f_\theta^1(x) + \cdots + \epsilon_x^k(\theta)\nabla_x f_\theta^k(x), \tag{15}$$

where the ϵ^i's are the signed residual, that is, the ith components of the signed error between the response function and the transformed target:

$$\epsilon_x^i(\theta) = a^i(f_\theta(x)) - y^i. \tag{16}$$

This means that the square norm of these gradients can be written as

$$\|\nabla_\theta L(x, y, \theta)\|^2 = \sum_i \epsilon_x^i(\theta)^2 \|\nabla_\theta f_\theta^i(x)\|^2 + A_\theta(x, y, \theta) \tag{17}$$

$$\|\nabla_x L(x, y, \theta)\|^2 = \sum_i \epsilon_x^i(\theta)^2 \|\nabla_x f_\theta^i(x)\|^2 + A_x(x, y, \theta), \tag{18}$$

where A_θ and A_x are the gradient alignment terms:

$$A_\theta(x, y, \theta) = \sum_{i \neq j} \langle \epsilon_x^i \nabla_\theta f_\theta^i(x), \epsilon_x^j \nabla_\theta f_\theta^j(x)\rangle \tag{19}$$

$$A_x(x, y, \theta) = \sum_{i \neq j} \langle \epsilon_x^i \nabla_x f_\theta^i(x), \epsilon_x^j \nabla_x f_\theta^j(x)\rangle. \tag{20}$$

A.1.1 Multi-class classification

Consider the multinoulli distribution where a random variable Z can take values in k classes, say $z \in \{1, \ldots, k\}$ with probabilities $p_1, \ldots, p_k$ for each class respectively. Let the transformed target map $y = \phi(z)$ associate to a class i its one-hot-encoded vector y with the ith component equal to 1 and all other components zero. The multinoulli density can then be written as $P(y) = p_1^{y_1} \cdots p_k^{y_k}$ which can be re-parameterized, showing that the multinoulli is a member of the exponential family distribution:

$$\begin{aligned}
P(y) &= \exp(\log(p_1^{y_1} \cdots p_l^{y_l})) \\
&= \exp\left(y_1 \log p_1 + \cdots + y_k \log p_k\right) \\
&= \exp\left(y_1 \log p_1 + \cdots + (1 - \sum_{l=1}^{k-1} y_l) \log p_k\right) \\
&= \exp\left(y_1 \log \frac{p_1}{p_k} + \cdots + y_{k-1} \log \frac{p_{k-1}}{p_k} + \cdots + \log p_k\right)
\end{aligned}$$

We can now express the natural parameter vector η in terms of the class probabilities:

$$\eta_i := \log \frac{p_i}{p_k} \quad \text{for } i = 1, 2, \ldots, k. \tag{21}$$

Note, $\eta_k = 0$. Taking the exponential of that last equation, and summing up, we obtain that $1/p_k = \sum_{i=1}^k e^{\eta_i}$, which we use to express the class probabilities in terms of the canonical parameters:

$$p_i = \frac{e^{\eta_i}}{\sum_l e^{\eta_l}}. \tag{22}$$

Since $1/p_k = \sum_{i=1}^k e^{\eta_i}$, we obtain that the log-partition function is

$$S(\eta) = \log \sum_{i=1}^k e^{\eta_i}, \tag{23}$$

whose derivative is the softmax function:

$$a(\eta) = \nabla_\eta S(\eta) = \frac{e^\eta}{\sum_{i=1}^k e^{\eta_i}}. \tag{24}$$

Now, if we estimate the natural parameter η with a neural network $\eta = f_\theta(x)$, we obtain the response

$$E(Y \mid x, \theta) = \nabla_\eta S(f_\theta(x)) = a(f_\theta(x)), \tag{25}$$

A.2 Computing the expanded modified loss from Section 5

In this section we verify that expanding the modified loss in Step 1. of Section 5; i.e., $\widetilde{L}_B = L_B + \frac{h}{4}\|\nabla L_B\|^2$, yields Eqn.(6). Namely, we show that

$$\widetilde{L}_B = L_B + \frac{h}{4B}\left(\frac{1}{B}\sum_{x,i} \epsilon_x^i(\theta)^2 \|\nabla_\theta f_\theta^i(x)\|^2\right) + \frac{h}{4}A_B(\theta) + \frac{h}{4}C_B(\theta), \tag{26}$$

where

$$A_B(\theta) = \frac{1}{B^2}\sum_{x\in B}\sum_{i\neq j}\langle \epsilon_x^i \nabla_\theta f_\theta^i(x), \epsilon_x^j \nabla_\theta f_\theta^j(x)\rangle \tag{27}$$

$$C_B(\theta) = \frac{1}{B^2}\sum_{(x,y)\neq(x',y')} \langle \nabla_\theta L(x,y,\theta), \nabla_\theta L(x',y',\theta)\rangle. \tag{28}$$

It suffices to show

$$\|\nabla_\theta L_B(\theta)\|^2 = \frac{1}{B^2}\sum_{x,i} \epsilon_x^i(\theta)^2 \|\nabla_\theta f_\theta^i(x)\|^2 \tag{29}$$

$$+ \frac{1}{B^2}\sum_{x\in B}\sum_{i\neq j}\langle \epsilon_x^i \nabla_\theta f_\theta^i(x), \epsilon_x^j \nabla_\theta f_\theta^j(x)\rangle \tag{30}$$

$$+ \frac{1}{B^2}\sum_{(x,y)\neq(x',y')} \langle \nabla_\theta L(x,y,\theta), \nabla_\theta L(x',y',\theta)\rangle. \tag{31}$$

Indeed this follows directly by computation. Firstly, note that

$$\begin{aligned}
\|\nabla_\theta L_B(\theta)\|^2 &= \left\langle \nabla_\theta \frac{1}{B}\sum_{x\in B} L(x,y,\theta), \nabla_\theta \frac{1}{B}\sum_{x'\in B} L(x',y',\theta)\right\rangle && (32)\\
&= \frac{1}{B^2}\sum_{x\in B}\langle \nabla_\theta L(x,y,\theta), \nabla_\theta L(x,y,\theta)\rangle && (33)\\
&\quad + \frac{1}{B^2}\sum_{(x,y)\neq(x',y')}\langle \nabla_\theta L(x,y,\theta), \nabla_\theta L(x',y',\theta)\rangle && (34)\\
&= \frac{1}{B^2}\sum_{x\in B}\|\nabla_\theta L(x,y,\theta)\|^2 && (35)\\
&\quad + \frac{1}{B^2}\sum_{(x,y)\neq(x',y')}\langle \nabla_\theta L(x,y,\theta), \nabla_\theta L(x',y',\theta)\rangle\,. && (36)
\end{aligned}$$

Now, by (17) and (19), the last equality can be simplified so that

$$\begin{aligned}
\|\nabla_\theta L_B(\theta)\|^2 &= \frac{1}{B^2}\sum_{x\in B}\sum_i \epsilon_x^i(\theta)^2\|\nabla_\theta f_\theta^i(x)\|^2 && (37)\\
&\quad + \frac{1}{B^2}\sum_{x\in B}\sum_{i\neq j}\langle \epsilon_x^i\nabla_\theta f_\theta^i(x), \epsilon_x^j\nabla_\theta f_\theta^j(x)\rangle && (38)\\
&\quad + \frac{1}{B^2}\sum_{(x,y)\neq(x',y')}\langle \nabla_\theta L(x,y,\theta), \nabla_\theta L(x',y',\theta)\rangle\,. && (39)
\end{aligned}$$

This completes the computation.

A.3 The Transfer Theorem

To frame the statement and proof of the Transfer Theorem, let's begin by setting up and defining some notation. Consider a deep neural network $f_\theta : \mathbb{R}^d \to \mathbb{R}^k$ parameterized by θ and consisting of l layers stacked consecutively. We can express f_θ as

$$f_\theta(x) = f_l \circ f_{l-1} \circ \cdots \circ f_1(x), \tag{40}$$

where $f_i(z) = a_i(w_i z + b_i)$ denotes the i-th layer of the network defined by the weight matrix w_i, the bias b_i and the layer activation function a_i which acts on the output z of the previous layer. In this way, we can write $\theta = (w_1, b_1, w_2, b_2, \ldots, w_l, b_l)$ to represent all the learnable parameters of the network.

Next, let $h_i(x)$ denote the sub-network from the input $x \in \mathbb{R}^d$ up to and including the output of layer i; that is,

$$h_i(x) = a_i(w_i h_{i-1}(x) + b_i), \text{ for } i = 1, 2, \ldots, l \tag{41}$$

where we understand $h_0(x)$ to be x. Note that, by this notation, $h_l(x)$ represents the full network; i.e., $h_l(x) = f_\theta(x)$.

At times it will be convenient to consider $f_\theta(x)$ as dependent only on a particular layer's parameters, for example w_i and b_i, and independent of all other parameter values in θ. In this case, we will use the notation f_{w_i} or f_{b_i} to represent f_θ as dependent only on w_i or b_i (resp.). Using the notation above, for each weight matrix $w_1, w_2, \ldots, w_l$, we can rewrite $f_{w_i}(x)$ as (similarly, for $f_{b_i}(x)$)

$$f_{w_i}(x) = g_i(w_i h_{i-1}(x) + b_i), \text{ for } i = 1, 2, \ldots, l \tag{42}$$

where each $g_i(z)$ denotes the remainder of the full network function $f_\theta(x)$ following the i-th layer. That is to say, g_i represents the part of the network deeper than the i-th layer (i.e., from layer i to the output $f_\theta(x)$) and h_i represents the part of the network shallower than the i-th layer (i.e., from the input x up to layer i).

We are now ready to state our main Theorem:

Theorem A.1 (Transfer Theorem)**.** *Consider a network $f_\theta : \mathbb{R}^d \to \mathbb{R}^k$ with l layers parameterized by $\theta = (w_1, b_1, \ldots, w_l, b_l)$, then we have the following inequality*

$$\|\nabla_x f_\theta(x)\|_F^2 \leq \frac{\|\nabla_\theta f_\theta(x)\|_F^2}{T_1^2(x,\theta) + \cdots + T_l^2(x,\theta)}, \tag{43}$$

where $T_i(x,\theta)$ is the transfer function for layer i given by

$$T_i(x,\theta) = \frac{1}{\sqrt{\min(d,k)}} \frac{\sqrt{1 + \|h_{i-1}(x)\|_2^2}}{\sigma_{\max}(w_i)\sigma_{\max}(\nabla_x h_{i-1}(x))}, \tag{44}$$

where h_i is the subnetwork to layer i and $\sigma_{\max}(A)$ is the maximal singular value of matrix A (i.e., its spectral norm).

The proof of Theorem A.1, inspired by the perturbation argument in [79], follows from the two following Lemmas. The idea is to examine the layer-wise structure of the neural network to compare the gradients $\nabla_{w_i} f_\theta$ with the gradients $\nabla_x f_\theta$. Due to the nature of $f_\theta(x)$ and its dependence on the inputs x and parameters θ, we show that at each layer i, a small perturbation of the inputs x of the model function $f_\theta(x)$ transfers to a small perturbation of the weights w_i.

Lemma A.2. *Let $f_\theta : \mathbb{R}^d \to \mathbb{R}^k$ represent a deep neural network consisting of l consecutive dense layers. Using the notation above, for $i = 1, 2, \ldots, l$, we have*

$$\|\nabla_x f_\theta(x)\|_2^2 \left(\frac{\|h_{i-1}(x)\|_2}{\|w_i\|_2 \|\nabla_x h_{i-1}(x)\|_2} \right)^2 \leq \|\nabla_{w_i} f_\theta(x)\|_2^2, \tag{45}$$

where $\|\cdot\|_2$ denotes the L^2 operator norm when applied to matrices and the L^2 norm when applied to vectors.

Proof. For $i = 1, 2, \ldots, l$ and following the notation above, the model function as it depends on the weight matrix w_i of layer i can be written as $f_{w_i}(x) = g_i(w_i h_{i-1}(x) + b_i)$. Now consider a small perturbation $x + \delta x$ of the input x. There exists a corresponding perturbation $w_i + u(\delta x)$ of the weight matrix in layer i such that

$$f_{w_i}(x + \delta x) = f_{w_i + u(\delta x)}(x). \tag{46}$$

To see this, note that for sufficiently small δx we can identify $h_i(x + \delta x)$ with its linear approximation around x so that $h_i(x + \delta x) = h_i(x) + \nabla_x h_i(x)\delta x$, where $\nabla_x h_i : \mathbb{R}^d \to \mathbb{R}^{|h_i|}$ is the total derivative of $h_i : \mathbb{R}^d \to \mathbb{R}^{|h_i|}$. Thus, by representing f_{w_i} as in (42), we have

$$\begin{aligned} f_{w_i}(x + \delta x) &= g_i(w_i h_{i-1}(x + \delta x) + b_i) \\ &= g_i(w_i h_{i-1}(x) + w_i \nabla_x h_{i-1}(x)\delta x + b_i). \end{aligned}$$

Similarly for $f_{w_i + u(\delta x)}(x)$, we have

$$\begin{aligned} f_{w_i + u(\delta x)}(x) &= g_i((w_i + u(\delta x))h_{i-1}(x) + b_i) \\ &= g_i(w_i h_{i-1}(x) + u(\delta x)h_{i-1}(x) + b_i). \end{aligned}$$

Thus Eqn. (46) is satisfied provided $u(\delta x)h_{i-1}(x) = w_i \nabla_x h_{i-1}(\delta x)$. Using the fact that $h_i(x)h_i(x)^T = \|h_i(x)\|_2^2$ and rearranging terms, we get

$$u(\delta x) = \frac{w_i(\nabla_x h_{i-1}(x)\delta x)h_{i-1}(x)^T}{\|h_{i-1}(x)\|_2^2}. \tag{47}$$

Defining $u(\delta x)$ as in Eqn. (47) and taking the derivative of both sides of Eqn. (46) with respect to δx at $\delta x = 0$ via the chain rule, we get (since by Eqn. (47) we have $u(\delta x)$ is linear in δx)

$$\nabla_x f_{w_i}(x) = \nabla_{w_i} f_\theta(x) \frac{(w_i \nabla_x h_{i-1}(x))h_{i-1}^T(x)}{\|h_{i-1}(x)\|_2^2}. \tag{48}$$

Finally, taking the square of the L^2 operator norm $\|\cdot\|_2$ on both sides, we have

$$\begin{aligned}\|\nabla_x f_{w_i}(x)\|_2^2 &= \left\|\nabla_{w_i} f_\theta(x)\frac{w_i\nabla_x h_{i-1}(x)h_{i-1}^T(x)}{\|h_{i-1}(x)\|_2^2}\right\|_2^2 \\ &\leq \|\nabla_{w_i} f_\theta(x)\|_2^2\frac{\|w_i\|_2^2\|\nabla_x h_{i-1}(x)\|_2^2}{\|h_{i-1}(x)\|_2^2}.\end{aligned}$$

Rearranging the terms and since $\nabla_x f_{w_i} = \nabla_x f_\theta$ we get (45) which completes the proof. □

Following the same argument, we can also prove the corresponding lemma with respect to the derivative of the biases b_i at each layer:

Lemma A.3. *Let $f_\theta : \mathbb{R}^d \to \mathbb{R}^k$ represent a deep neural network consisting of l consecutive dense layers. Using the notation above, for $i = 1, 2, \ldots, l$, we have*

$$\|\nabla_x f_\theta(x)\|_2^2\left(\frac{1}{\|w_i\|_2\|\nabla_x h_{i-1}(x)\|_2}\right)^2 \leq \|\nabla_{b_i} f_\theta(x)\|_2^2, \tag{49}$$

where $\|\cdot\|_2$ denotes the L^2 operator norm when applied to matrices and the L^2 norm when applied to vectors.

Proof. The proof is similar in spirit to the proof of Lemma A.2. Namely, for small perturbations $x + \delta x$ of the input x, we verify that we can find a corresponding small perturbation $b_i + u(\delta x)$ of the bias b_i so that $f_{b_i}(x + \delta x) = f_{b_i+u(\delta x)}(x)$. From Eqn. (42), this time in relation to the bias term, we can write $f_{b_i}(x) = g_i(w_i h_{i-1}(x) + b_i)$. Then, taking a small perturbation $x + \delta x$ of the input x and simplifying as before, we get

$$u(\delta x) = w_i\nabla_x h_{i-1}(x)\delta x. \tag{50}$$

As before, taking this as our definition of $u(\delta x)$, and taking the derivative w.r.t. $\delta(x)$ as before via the chain rule, we obtain

$$\nabla_x f_{b_i}(x) = \nabla_{b_i} f_\theta(x) w_i\nabla_x h_{i-1}(x). \tag{51}$$

Again, after taking the square of the L^2 operator norm on both sides, and moving the terms around we get

$$\|\nabla_x f_\theta(x)\|_2^2\left(\frac{1}{\|w_i\|_2\|\nabla_x h_{i-1}(x)\|_2}\right)^2 \leq \|\nabla_{b_i} f_\theta(x)\|_2^2. \tag{52}$$

□

It remains to prove Theorem A.1:

Proof of Theorem A.1. Recall, that $f_\theta(x)$ represents a deep neural network consisting of l layers and parameterized by $\theta = (w_1, b_1, w_2, b_2, \ldots, w_l, b_l)$. Furthermore, we use the notation f_{w_i} (resp. f_{b_i}) to denote f_θ as dependent only on w_i (resp. b_i); i.e., all other parameter values are considered constant. Let $\|\cdot\|_F$ denote the Frobenius norm. For this model structure, by Pythagoras's theorem, it follows that

$$\|\nabla_\theta f_\theta(x)\|_F^2 = \|\nabla_{w_1} f_\theta(x)\|_F^2 + \|\nabla_{b_1} f_\theta(x)\|_F^2 + \cdots + \|\nabla_{w_l} f_\theta(x)\|_F^2 + \|\nabla_{b_l} f_\theta(x)\|_F^2. \tag{53}$$

The remainder of the proof relies on the following general property of matrix norms: Given a matrix $A \in \mathbb{R}^{m\times n}$, let $\sigma_{\max}(A)$ represent the largest singular value of A. The Frobenius norm $\|\cdot\|_F$ and the L^2 operator norm $\|\cdot\|_2$ are related by the following inequalities

$$\|A\|_2^2 = \sigma_{\max}^2(A) \leq \|A\|_F^2 = \sum_{i=1}^{\min(m,n)} \sigma_k^2 \leq \min(m,n)\cdot\sigma_{\max}^2(A) = \min(m,n)\cdot\|A\|_2^2 \tag{54}$$

where $\sigma_i(A)$ are the singular values of the matrix A. Considering $\nabla_x f_\theta$ as a map from $\mathbb{R}^d$ to $\mathbb{R}^k$, by (54) it follows that

$$\|\nabla_x f_\theta(x)\|_2^2 \geq \frac{1}{\min(d,k)}\|\nabla_x f_\theta(x)\|_F^2. \tag{55}$$

By Lemma A.2 and Lemma A.3, we have that for $i = 1, 2, \ldots, l$

$$\|\nabla_{w_i} f_\theta(x)\|_2^2 + \|\nabla_{b_i} f_\theta(x)\|_2^2 \quad \geq \quad \|\nabla_x f_\theta(x)\|_2^2 \left(\frac{1 + \|h_{i-1}(x)\|_2^2}{\|w_i\|_2^2 \|\nabla_x h_{i-1}(x)\|_2^2} \right). \tag{56}$$

Therefore, we can re-write Eqn. (53) using (54), (55) and (56) to get

$$\begin{aligned}
\|\nabla_\theta f_\theta(x)\|_F^2 &= \sum_{i=1}^{l} \left(\|\nabla_{w_i} f_\theta(x)\|_F^2 + \|\nabla_{b_i} f_\theta(x)\|_F^2 \right) \\
&\geq \sum_{i=1}^{l} \|\nabla_{w_i} f_\theta(x)\|_2^2 + \|\nabla_{b_i} f_\theta(x)\|_2^2 \\
&\geq \sum_{i=1}^{l} \|\nabla_x f_\theta(x)\|_2^2 \left(\frac{1 + \|h_{i-1}(x)\|_2^2}{\|w_i\|_2^2 \|\nabla_x h_{i-1}(x)\|_2^2} \right) \\
&\geq \|\nabla_x f_\theta(x)\|_F^2 \sum_{i=1}^{l} \frac{1}{\min(d,k)} \left(\frac{1 + \|h_{i-1}(x)\|_2^2}{\sigma_{\max}^2(w_i) \sigma_{\max}^2(\nabla_x h_{i-1}(x))} \right).
\end{aligned}$$

For $i = 1, 2, \ldots, l$, define

$$T_i(x, \theta) := \frac{1}{\sqrt{\min(d,k)}} \frac{\sqrt{1 + \|h_{i-1}(x)\|_2^2}}{\sigma_{\max}(w_i) \sigma_{\max}(\nabla_x h_{i-1}(x))}. \tag{57}$$

We call $T_i(x, \theta)$ the *transfer function for layer* i in the network.

Then, rearranging terms in the final inequality above, we get

$$\|\nabla_x f_\theta(x)\|_F^2 \leq \frac{\|\nabla_\theta f_\theta(x)\|_F^2}{T_1^2(x,\theta) + \cdots + T_l^2(x,\theta)}. \tag{58}$$

This completes the proof. □

B Experiment details

B.1 Figure 1: Motivating 1D example

We trained a ReLU MLP with 3 layers of 300 neurons each to regress 10 data points on an exact parabola $(x, x^2) \in \mathbb{R}^2$ where x ranges over 10 equidistant points in the interval $[-1, 1]$. We use full batch gradient descent with learning rate 0.02 for 100000 steps/epochs. We plotted the model function across the $[-1, 1]$ range and computed its geometric complexity over the dataset at step 0 (initialization), step 10, step 1000, and step 10000 (close to interpolation). The network was randomly initialized using the standard initialization (i.e. the weights were sampled from a truncated normal distribution with variance inversely proportional to the number of input units and the bias terms were set to zero). This model was trained five separate times using five different random seeds. Each line marks the mean of the five runs and the shaded region is 95% confidence interval over these five seeds.

B.2 Figure 2: Geometric complexity and initialization

Left and Middle: We initialized several ReLU MLP's $f_{\theta_0} : \mathbb{R}^d \to \mathbb{R}^k$ with large input and output: $d = 224 \times 224 \times 3$ and $k = 1000$, with 500 neurons per layer, and with a varying number of layers $l \in [1, 2, 4, 8, 16, 32, 64]$. We used the standard initialization scheme: we sample them from a truncated normal distribution with variance inversely proportional to the number of input units and the bias terms were set to zero. We measured and plotted the following quantities

$$f_{\text{mean}}(x) = \text{mean}\left\{ f_{\theta_0}^i(P_1 + (P_2 - P_1)x), \quad i = 1, \ldots, 1000 \right\}, \tag{59}$$

$$f_{\max}(x) = \max\left\{ (|f_{\theta_0}^i(P_1 + (P_2 - P_1)x))|, \quad i = 1, \ldots, 1000 \right\}, \tag{60}$$

with x ranging over 50 equidistant points in the interval $[0, 1]$. The points P_1 and P_2 where chosen to be the two diagonal points $(-1, \ldots, -1)$ and $(1, \ldots, 1)$, respectively, of the normalized data

hyper-cube $[-1,1]^d$. Each line marks the mean of the 5 runs and the shaded region is 95% confidence interval over these five seeds.

What we observe with the standard initialization scheme also persists with the Glorot initialization scheme, where the biases are set to zero, and the weight matrices parameters are sampled from the uniform distribution on $[-1,1]$ and scaled at each layer by $\sqrt{6/d_l + d_{l-1}}$, where d_l is the number of units in layer l. We report this in Fig. 6 below.

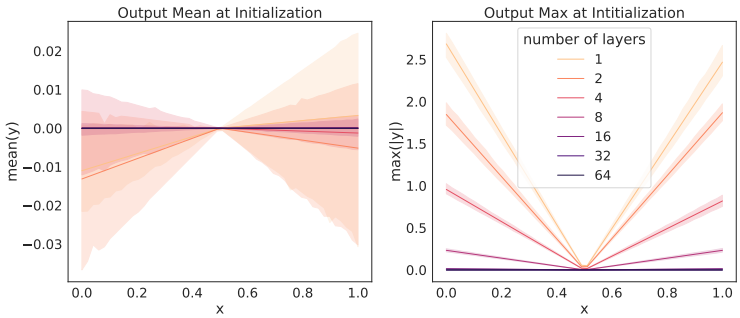

Figure 6: Additional plots to complement Fig. 2: Deeper neural Relu networks initialize closer to the zero function with the Glorot scheme on normalized data. We repeat the setup of Fig. 2 Left and Middle described in Section B.2 but with the Glorot scheme instead of the standard scheme.

Right: We measured the geometric complexity $\langle f_{\theta_0}, D\rangle_G$ at initialization for ReLU MLP's $f_{\theta_0} : \mathbb{R}^d \to \mathbb{R}^k$ with large input and output: $d = 224 \times 224 \times 3$ and $k = 1000$, with 500 neurons per layer, and with a varying number of layers $l \in [1,2,4,8,16,32,64]$. The geometric complexity was computed over a dataset D of 100 points sampled uniformly from the normalized data hyper-cube $[-1,1]^d$. For each combination of activation and initialization in $\{\text{ReLU}, \text{sigmoid}\} \times \{\text{standard}, \text{Glorot}\}$ we repeated the experiment 5 times with different random seeds. The Glorot initialization scheme is the one described in the paragraph above. For the standard initialization we set the biases to zero, and initialized the weight matrices parameters from a normal distribution truncated to the range $[-2,2]$ and rescaled using $1/\sqrt{d_{l-1}}$ at each layer. We plotted the mean geometric complexity with error bars representing the 95% confidence interval over the 5 random seeds. The error bars are tiny in comparison of the plotted quantities and are therefore not visible on the plot.

B.3 Figure 3: Geometric complexity and explicit regularization

Left: We trained a ResNet18 on CIFAR10 three times with different random seeds with a learning rate of 0.02, batch size of 512, for 10000 steps for each combination of regularization rate and regularization type in $\{\text{L2}, \text{Spectral}, \text{Flatness}\} \times [0, 0.01, 0.025, 0.05, 0.075, 0.1]$. We measured the geometric complexity at time of maximum test accuracy for each of these runs and plotted the mean with 95% confidence interval over the 3 random seeds. For the L2 regularization we added the sum of the parameter squares to the loss multiplied with the regularization rate. For the spectral regularization, we followed the procedure described in [90] by adding to the loss the penalty $(\alpha/2)\sum_i \sigma_{\max}(W_i)^2$ where W_i is either the layer weight matrix for a dense layer or, for a convolution layer, the matrix of shape $b \times ak_wk_h$ obtained by reshaping the convolution layer with a input channels, b output channels, and a kernel of size $k_w \times k_h$. For the flatness regularization, we added to the batch loss L_B at each step the norm square of the batch loss gradient $\|\nabla_\theta L_B(\theta)\|^2$ multiplied by the regularization rate.

Middle: We trained a ResNet18 on CIFAR10 with learning rate 0.005 and batch size 16 for 100000 steps with varying proportion of label noise. For each label noise proportion $\alpha \in [0, 0.05, 0.1, 0.15, 0.25]$, we trained three times with different random seeds. We plotted the mean geometric complexity at time of maximum test accuracy as well as the 95% confidence interval over the three runs. The label noise was created by mislabelling α % of the true labels before training.

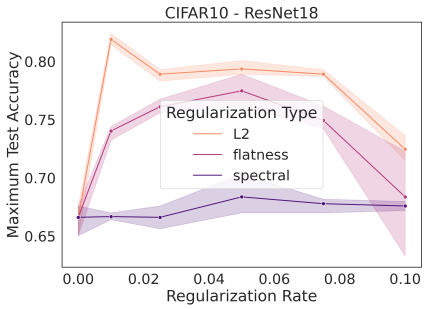

Figure 7: Additional plot to complement Fig. 3: Maximum test accuracy recorded for different levels of explicit regularization.

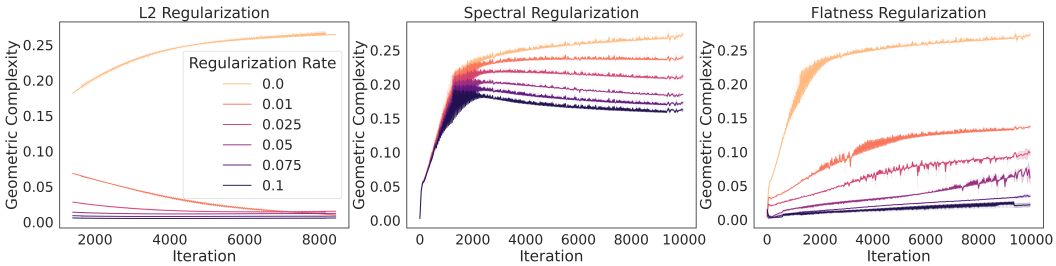

Figure 8: Additional plots to complement Fig. 3: GC plotted against training iterations for different explicit regularization types and rates for ResNet18 trained on CIFAR10.

Right: We trained a MLP with six layers of 214 neurons on MNIST with learning rate 0.005 and batch size 32 for 128000 steps with varying proportion of label noise. For each label noise proportion $\alpha \in [0, 0.05, 0.1, 0.15, 0.25]$, we trained the neural network 5 times with a different random seed. We plotted the mean geometric complexity at time of maximum accuracy as well as the 95% confidence interval over the 5 runs. The label noise was created by mislabelling α % of the true labels before training.

B.4 Figure 4: Geometric complexity and implicit regularization

Top row (varying learning rates): We trained a ResNet18 on CIFAR10 with batch size 512, for 30000 steps with varying learning rates. For each learning rate in $[0.005, 0.01, 0.05, 0.1, 0.2]$, we trained the neural network three times with a different random seed. We plotted the learning curves for the test accuracy, the geometric complexity, and the training loss, where the solid lines represent the mean of these quantities over the three runs and the shaded area represents the 95% confidence interval over the three runs. We applied a smoothing over 50 steps for each run, before computing the mean and the confidence interval of the three runs.

Bottom row (varying batch sizes): We trained a ResNet18 on CIFAR10 with learning rate 0.2, for 100000 steps with varying batch sizes. For each batch size in $[8, 16, 32, 64, 128, 256, 512, 1024]$, we trained the neural network three times with a different random seed. We plotted the learning curves for the test accuracy, the geometric complexity, and the training loss, where the solid lines represent the mean of these quantities over the three runs and the shaded area represents the 95% confidence interval over the three runs. We applied a smoothing over 50 steps for each run, before computing the mean and the confidence interval of the three run.

B.5 Figure 5: Geometric complexity and double-descent

We train a ResNet18 on CIFAR10 with learning rate 0.8, batch size 124, for 100000 steps, with 18 different network widths. For each network width in $[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 20, 30, 64, 100, 128]$ we train three times with a different random seed.

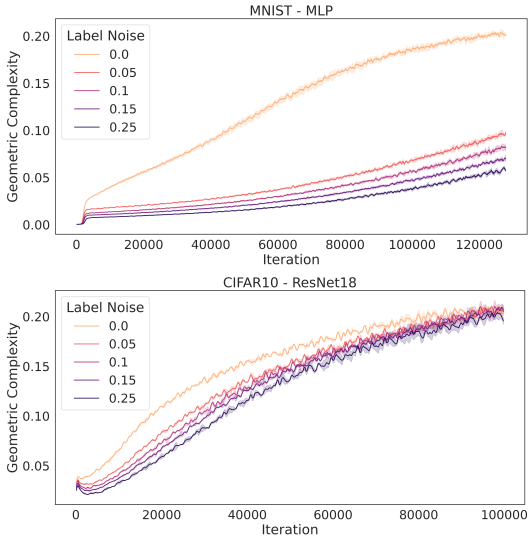

Figure 9: Additional plots to Fig. 3 experiments: GC plotted against training iterations for different label noise proportions.

Left: We measure the geometric complexity as well as the test loss at the end of training. We plot the mean and the 95% confidence interval of both quantities over the three runs against the network width. The critical region in yellow identified experimentally in [71] indicates the transition between the under-parameterized regime and the over-parameterized regime, where the second descent starts.

Right: We plot the test loss measured at the end of training against the geometric complexity at the end of training. The top plot shows every model width for every seed, while the bottom plot shows averages of these quantities over the three seeds. We then fit the data with a polynomial of degree six to sufficiently capture any high order relationship between the test loss and geometric complexity.

Definition of width: We follow the description of ResNet width discussed in [71]. The ResNet18 architecture we follow is that of [38]. The original ResNet18 architecture has four successive ResNet blocks each formed by two identical stacked sequences of a BatchNorm and a convolution layer. The number of filters for the convolution layers in each of the successive ResNet block is $(k, 2k, 4k, 8k)$ with $k = 64$ for the original RestNet18. Following [71], we take k to be the width of the network, which we vary during our experiments.

C Additional Experiments

C.1 Geometric complexity at initialization decreases with added layers on large domains

We reproduce the experiments from Fig. 2 on a larger domain with the same conclusion: For both the standard and the Glorot initialization schemes ReLU networks initialize closer to the zero function and with lower geometric complexity as the number of layer increases.

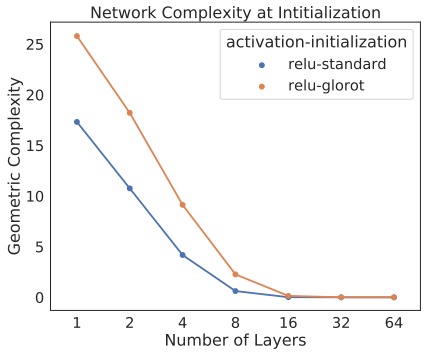

Figure 10: **Geometric complexity at initialization decreases with the number of layers up to zero even on large domains:** We repeat the setup of Fig. 2 Right described in Section B.2 but we sample the dataset from points in a large domain $[-1000, 1000]^d$ instead of the normalized hyper-cube $[-1, 1]^d$. We measure the geometric complexity for both the Glorot and the standard initialization schemes.

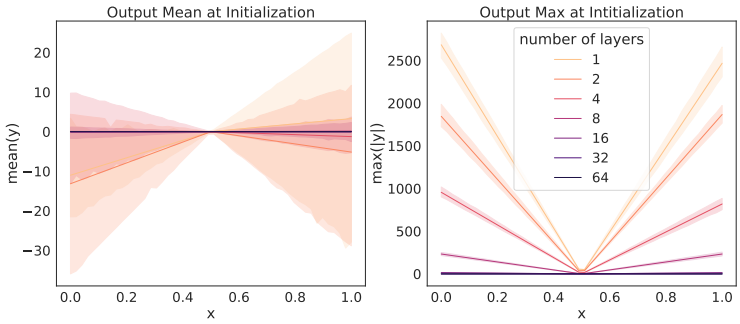

Figure 11: **Deeper neural Relu networks initialize closer to the zero function with the Glorot scheme even on large domains:** We repeat the setup of Fig. 2 Left and Middle described in Section B.2 but we evaluate the networks on a diagonal of the larger hyper-cube $[-1000, 1000]^d$ instead of the normalized hyper-cube $[-1, 1]^d$.

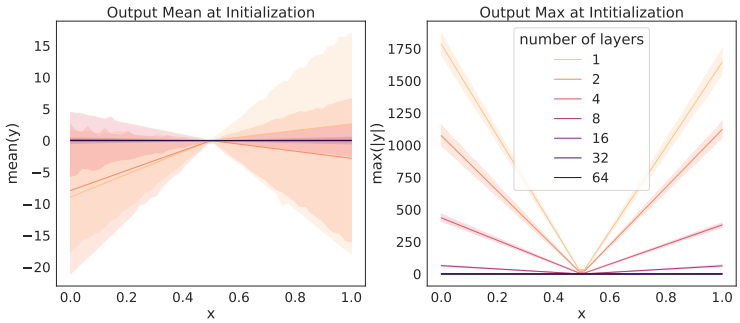

Figure 12: **Deeper neural Relu networks initialize closer to the zero function with the standard scheme even on large domains:** We repeat the setup of Fig. 2 Left and Middle described in Section B.2 but we evaluate the networks on a diagonal of the larger hyper-cube $[-1000, 1000]^d$ instead of the normalized hyper-cube $[-1, 1]^d$.

C.2 Geometric complexity decreases with implicit regularization for MNIST

We replicate the implicit regularization experiments we conducted for CIFAR10 with ResNet18 in Fig. 4 for MLP's trained on MNIST. The conclusion is the same: higher learning rates and smaller batch sizes decrease geometric complexity through implicit gradient regularization, and are correlated with higher test accuracy.

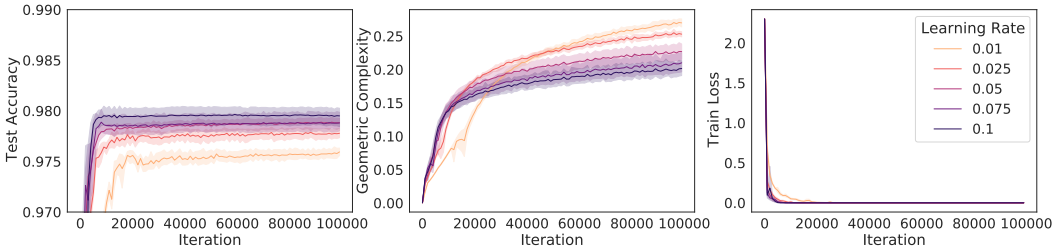

Figure 13: **Geometric complexity decreases with higher learning rates on MNIST**: We trained a selection of MLP's with 6 hidden layers with 500 neurons per layer on MNIST with batch size of 512, for 100000 steps and with varying batch sizes. For each learning rate in $[0.01, 0.025, 0.05, 0.075, 0.1]$, we trained 5 different times with a different random seed. The MLP were initialized using the standard initialization scheme.

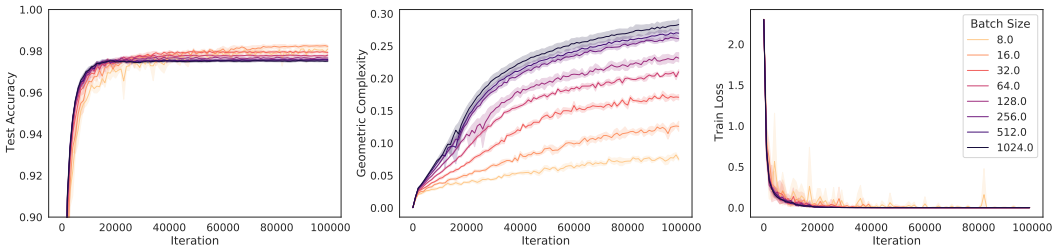

Figure 14: **Geometric complexity decreases with smaller batch sizes on MNIST**: We trained a selection of MLP's with 6 hidden layers with 500 neurons per layer on MNIST with learning rate of 0.02, for 100000 steps and with varying batch sizes. For each batch size in $[8, 16, 32, 64, 128, 256, 512, 1024]$, we trained 5 different times with a different random seed. The MLP were initialized using the standard initialization scheme.

C.3 Geometric complexity decreases with explicit regularization for MNIST

We replicate the explicit regularization experiments we conducted for CIFAR10 with ResNet18 in Fig. 3 for MLP's trained on MNIST with similar conclusions: higher regularization rates for L2, spectral, and flatness regularization decrease geometric complexity and are correlated with higher test accuracy.

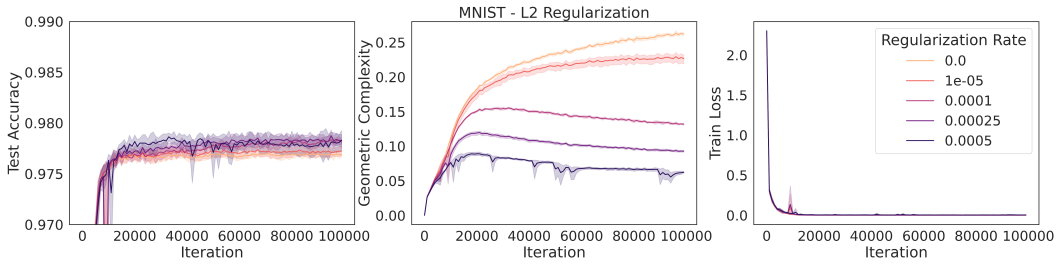

Figure 15: **Geometric complexity decreases with L2 regularization on MNIST**: We trained a selection of MLP's with 6 hidden layers with 500 neurons per layer on MNIST with learning rate of 0.02, batch size of 512, for 100000 steps. We regularized the loss by adding to it the L2 norm penalty $\alpha \sum_i \|W_i\|_F^2$ where W_i are the layer weight matrices. For each regularization rate $\alpha \in [0, 0.00001, 0.0001, 0.00025, 0.0005]$, we trained 5 different times with a different random seed. The MLP were initialized using the standard initialization scheme.

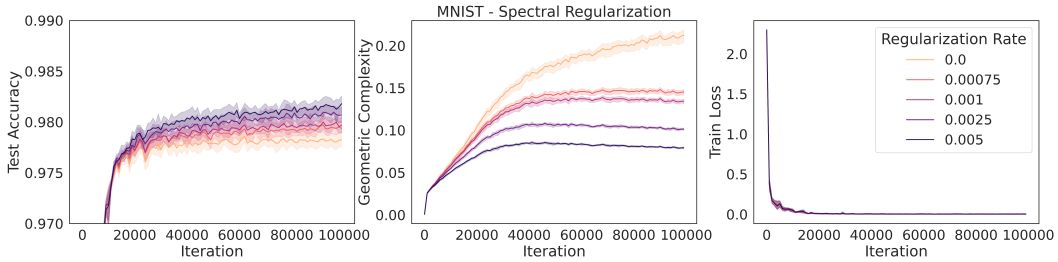

Figure 16: **Geometric complexity decreases with spectral regularization on MNIST**: We trained a selection of MLP's with 4 hidden layers with 200 neurons per layer on MNIST with learning rate of 0.02, batch size of 128, for 100000 steps. We regularized the loss by adding to it the spectral norm penalty $\alpha/2 \sum_i \sigma_{\max}(W_i)^2$ where W_i are the layer weight matrices as described in [67]. For each regularization rate $\alpha \in [0, 0.00075, 0.001, 0.0025, 0.005]$, we trained 5 different times with a different random seed. The MLP were initialized using the standard initialization scheme.

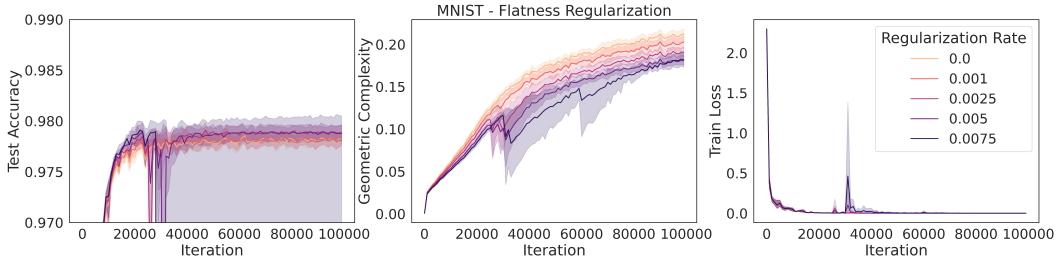

Figure 17: **Geometric complexity decreases with flatness regularization on MNIST**: We trained a selection of MLP's with 4 hidden layers with 200 neurons per layer on MNIST with learning rate of 0.02, batch size of 128, for 100000 steps. We regularized the loss by adding to it the gradient penalty $\alpha \|\nabla_\theta L_B(\theta)\|^2$ where L_B is the batch loss. For each regularization rate $\alpha \in [0, 0.001, 0.0025, 0.005, 0.0075]$, we trained 5 different times with a different random seed. The MLP were initialized using the standard initialization scheme.

C.4 Explicit geometric complexity regularization for MNIST and CIFAR10

In this section, we explicitly regularize for the geometric complexity. This is a known form of regularization also called Jacobian regularization [42, 84, 89, 90]. We first perform this regularization for a MLP trained on MNIST (Fig. 18) and then for a ResNet18 trained on CIFAR10 (Fig. 19) with the following conclusion: test accuracy increases with higher regularization strength while the geometric complexity decreases.

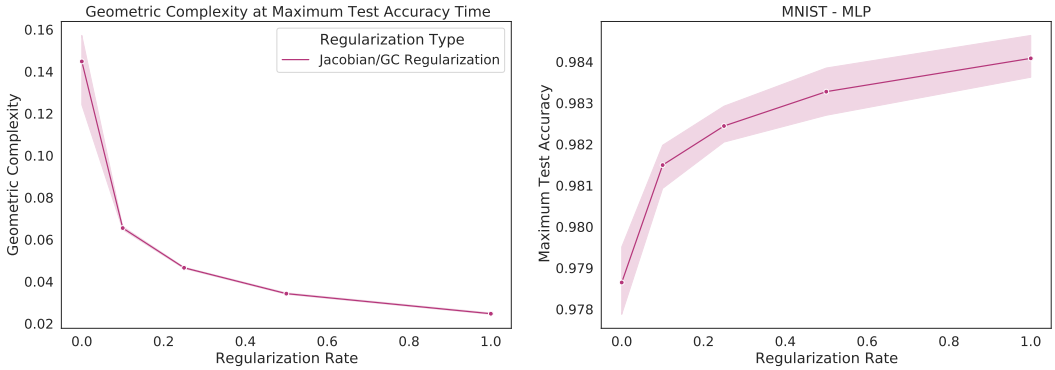

Figure 18: **Test accuracy increases with explicit GC regularization on MNIST**: We trained a selection of MLP's with 4 hidden layers with 200 neurons per layer on MNIST with learning rate of 0.02, batch size of 128, for 100000 steps. We regularized the loss by adding to it the gradient penalty $\alpha/B \sum_{x \in B} \|\nabla_x f_\theta(x)\|_F^2$ where $f_\theta(x)$ is the logit network. For each regularization rate $\alpha \in [0, 0.1, 0.25, 0.5, 1]$, we trained 5 different times with a different random seed. The MLP were initialized using the standard initialization scheme.

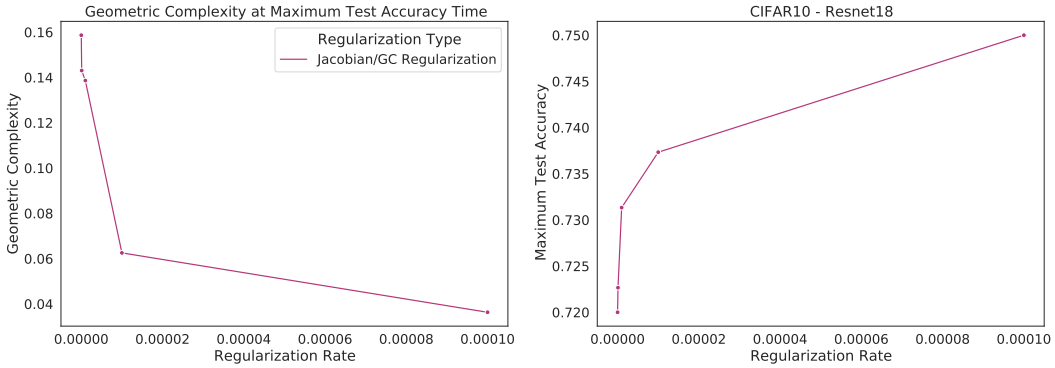

Figure 19: **Test accuracy increases with explicit GC regularization on CIFAR10**: We trained a selection of ResNet18 with learning rate of 0.02, batch size of 128, for 10000 steps. We regularized the loss by adding to it the gradient penalty $\alpha/B \sum_{x \in B} \|\nabla_x f_\theta(x)\|_F^2$ where $f_\theta(x)$ is the logit network. For each regularization rate $\alpha \in [0, 0.0000001, 0.000001, 0.00001, 0.0001]$, we trained only one time with a single random seed, and the training had to be stopped before reaching peak test accuracy because of the heavy computational time due to this regularization.

C.5 Separate L2, flatness, and spectral regularization experiments for CIFAR10

For the sake of space in Fig. 3 (right) in the main paper, we used the same regularization rate range for all types of explicit regularization we tried. In this section, we perform the experiments on a targeted range for each regularization type, leading to clearer plots (Fig. 20, Fig. 21, and Fig. 22).

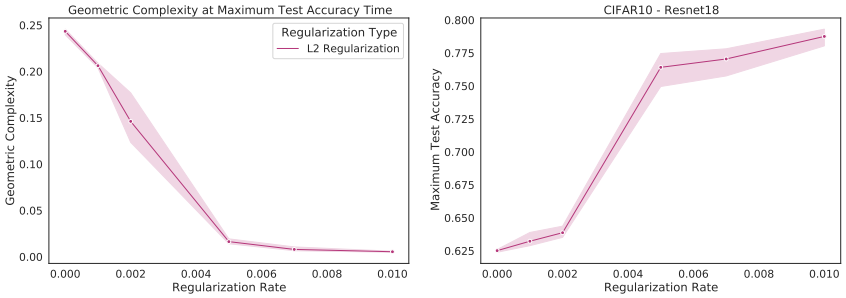

Figure 20: **GC decreases with explicit L2 regularization on CIFAR10**: We trained a selection of ResNet18 with learning rate of 0.02, batch size of 512, for 10000 steps. We regularized the loss by adding to it the standard L2 loss penalty. For each regularization rate $\alpha \in [0, 0.001, 0.002, 0.005, 0.007, 0.01]$, we trained 3 different times with a different random seed.

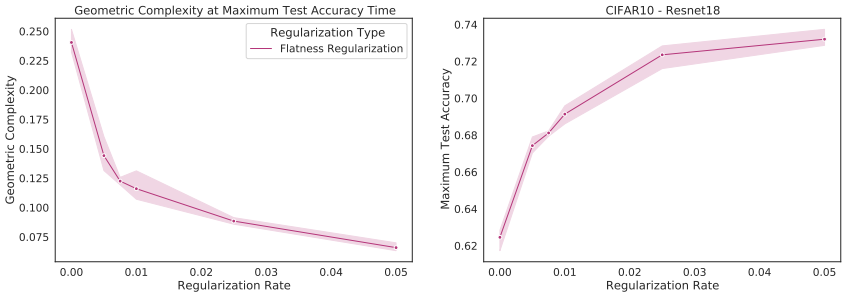

Figure 21: **GC decreases with explicit flatness regularization on CIFAR10**: We trained a selection of ResNet18 with learning rate of 0.02, batch size of 512, for 10000 steps. We regularized the loss by adding to it the gradient penalty $\alpha\|\nabla_\theta L_B(\theta)\|^2$ where L_B is the batch loss. For each regularization rate $\alpha \in [0, 0.005, 0.0075, 0.01, 0.025, 0.05]$, we trained 3 different times with a different random seed.

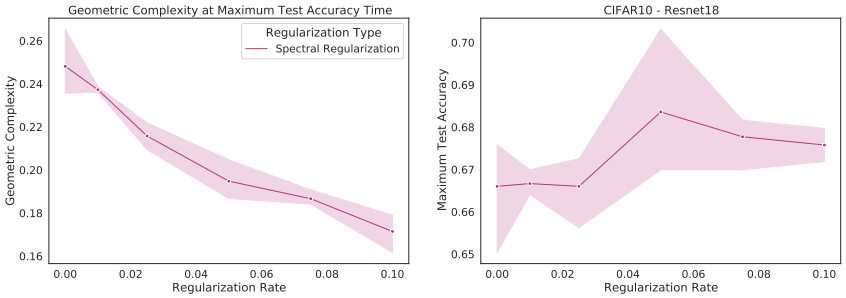

Figure 22: **Geometric complexity decreases with spectral regularization on CIFAR10:** We trained a selection of ResNet18 on CIFAR10 with learning rate of 0.02, batch size of 512, for 100000 steps. For each regularization rate $\alpha \in [0, 0.01, 0.025, 0.05, 0.075, 0.1]$, we trained 3 different times with a different random seed.

C.6 Geometric complexity in the presence of multiple tuning mechanisms

For most of this paper, we studied the impact of tuning strategies, like the choice of initialization, hyper-parameters, or explicit regularization in isolation from other very common heuristics like learning rate schedules and data-augmentation. In this section, we reproduce the implicit regularization effect of the batch size and the learning rate on GC (c.f. Fig. 4 in the main paper) while using these standard tricks to achieve better performance. The resulting models achieve performance closer to SOTA for the ResNet18 architecture.

Although the learning curves are messier and harder to interpret (Fig. 23) because of the multiple mechanisms interacting in complex ways, we still observe that the general effect of the learning rate (Fig. 24) and the batch size (Fig. 25) on geometric complexity is preserved in this context. More importantly, we also note that the sweeps with higher test accuracy solutions tend also to come with a lower geometric complexity, even in this more complex setting (Fig. 24 right and Fig. 25 right). Namely, models with higher test accuracy have correspondingly lower GC. Specifically in terms of implicit regularization, as the learning rate increases, the geometric complexity decreases and the maximum test accuracy increases. Similarly, smaller batch size leads to lower geometric complexity as well as higher test accuracy.

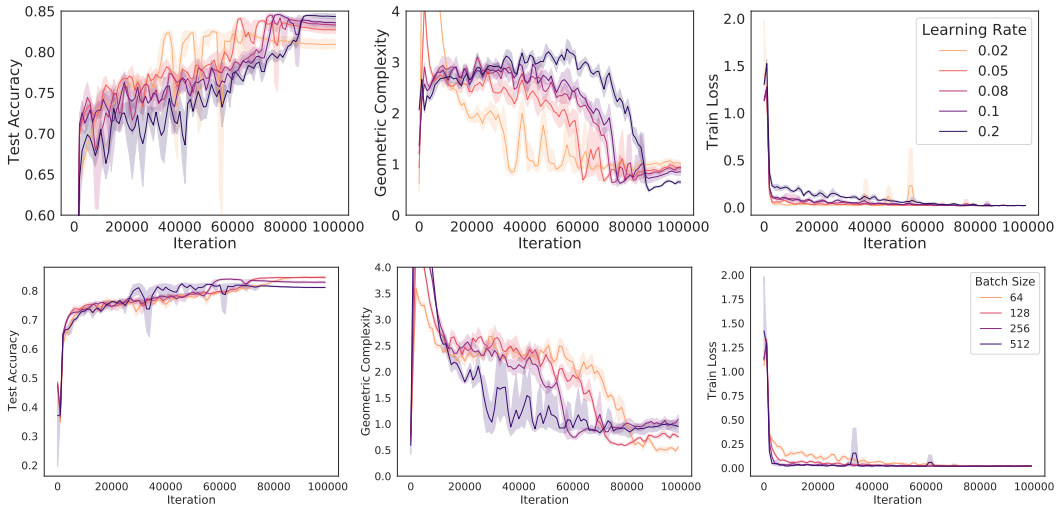

Figure 23: Impact of IGR when training ResNet18 on CIFAR10 with multiple tuning mechanisms including cosine learning rate scheduler, data augmentation and L2 regularization. Note that the GC is computed on batches of size 128 which leads to a lot of variance in the estimate. **Top row:** As IGR increases through higher learning rates, GC decreases. **Bottom row:** Similarly, lower batch size leads to decreased GC.

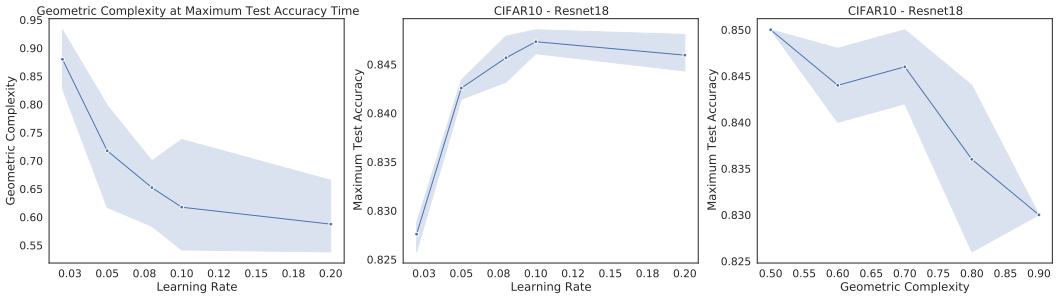

Figure 24: **GC decreases as learning rate and model test accuracy increases on CIFAR10**: We trained a collection of ResNet18 models on CIFAR10 with varying initial learning rates $h \in [0.02, 0.05, 0.08, 0.1, 0.2]$ and cosine learning rate schedule. Each job was trained with SGD without momentum for 100000 steps, with batch size 128 and L2 regularized loss with regularization rate 0.005. We also included data augmentation in the form of random flip. Test accuracy is reported as best test accuracy during training. GC is computed during training on the training batches, which produces a large variance in the estimate when the batch size is small.

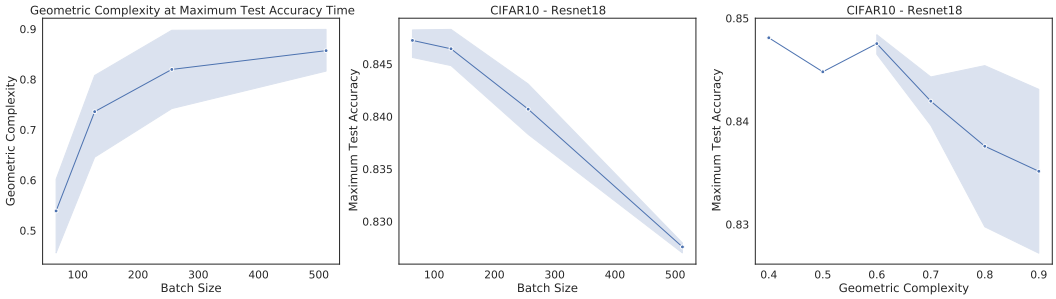

Figure 25: **GC increases as batch size increases on CIFAR10**: We trained a collection of ResNet18 models on CIFAR10 with varying batch sizes of 64, 128, 256, and 512. Each job was trained with SGD without momentum for 100000 steps, with cosine learning rate scheduler initialized at 0.02 and L2 regularized loss with regularization rate 0.005. We also included data augmentation in the form of random flip. Test accuracy is reported as best test accuracy during training. GC is computed during training on the training batches, which produces a large variance in the estimate when the batch size is small.

C.7 Geometric complexity in the presence of momentum

We replicate the implicit and explicit regularization experiments using SGD *with momentum*, which is widely used in practise. The conclusion remains the same as for vanilla SGD: More regularization (implicit through batch size or learning rate or explicit through flatness, spectral, and L2 penalties) produces solutions with higher test accuracy and lower geometric complexity.

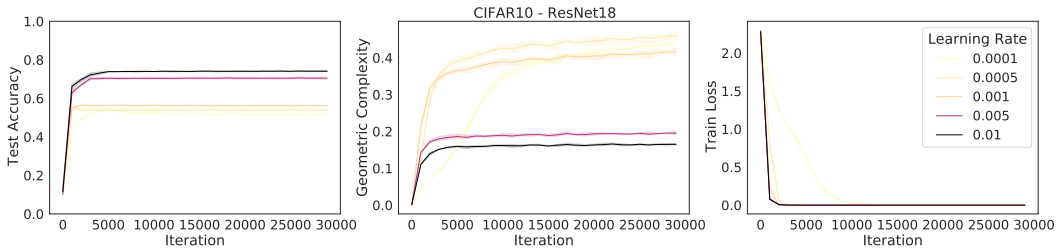

Figure 26: **Geometric complexity decreases with higher learning rates on CIFAR10 trained with momentum**: We trained a selection of ResNet18 with batch size of 512 for 30000 steps using SGD with a momentum of 0.9. For each learning rate $\alpha \in [0.0001, 0.0005, 0.001, 0.005, 0.01]$, we trained 3 different times with a different random seed.

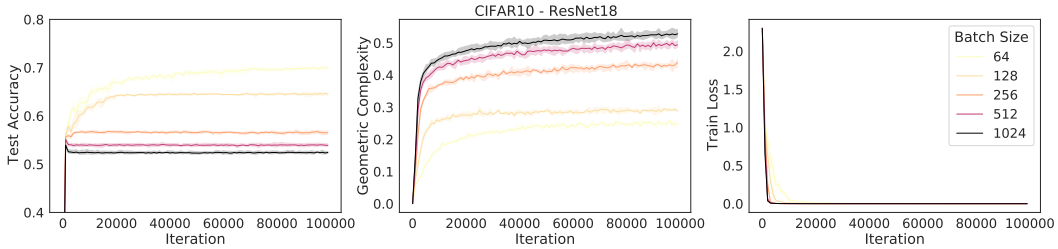

Figure 27: **Geometric complexity decreases with lower batch sizes on on CIFAR10 trained with momentum**: We trained a selection of ResNet18 with learning rate of 0.0005 for 100000 steps using SGD with a momentum of 0.9. For each batch size in $\alpha \in [64, 128, 256, 512, 1024]$, we trained 3 different times with a different random seed.

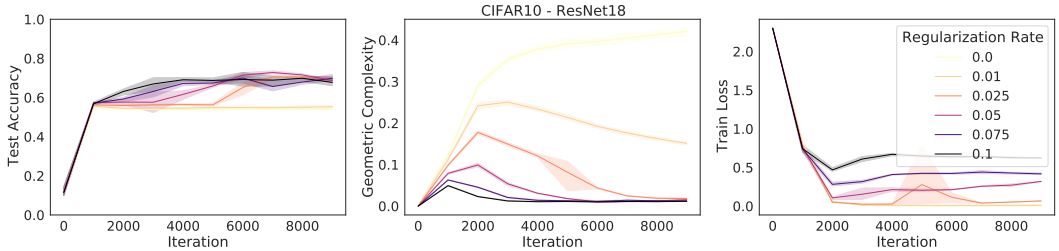

Figure 28: **Geometric complexity decreases with increased L2 regularization on CIFAR10 trained with momentum**: We trained a selection of ResNet18 with learning rate of 0.0005 with batch size of 512 for 10000 steps using SGD with a momentum of 0.9. For each regularization rate in $\alpha \in [0, 0.01, 0.025, 0.05, 0.075, 0.1]$, we trained 3 different times with a different random seed.

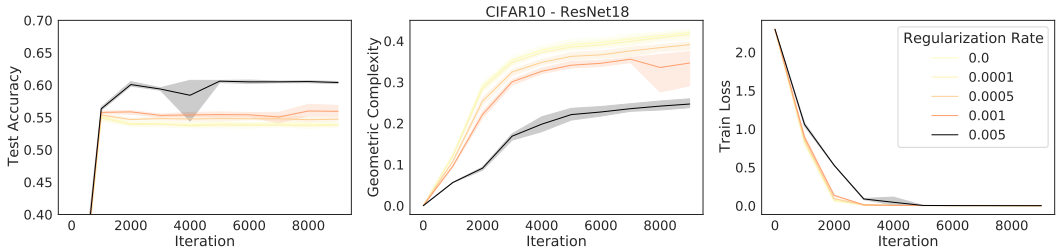

Figure 29: **Geometric complexity decreases with increased flatness regularization on CIFAR10 trained with momentum**: We trained a selection of ResNet18 with learning rate of 0.0005 with batch size of 512 for 10000 steps using SGD with a momentum of 0.9. We regularized the loss by adding to it the gradient penalty $\alpha\|\nabla_\theta L_B(\theta)\|^2$ where L_B is the batch loss. For each regularization rate in $\alpha \in [0, 0.0001, 0.0005, 0.001, 0.005]$, we trained 3 different times with a different random seed.

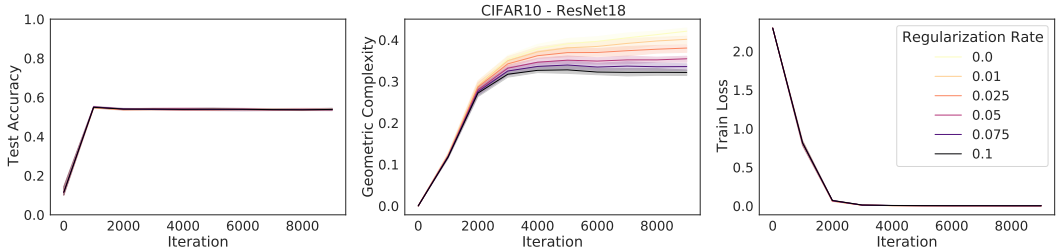

Figure 30: **Geometric complexity decreases with increased spectral regularization on CIFAR10 trained with momentum**: We trained a selection of ResNet18 with learning rate of 0.0005 with batch size of 512 for 10000 steps using SGD with a momentum of 0.9. We regularized the loss by adding to it the spectral norm penalty $\alpha/2 \sum_i \sigma_{\max}(W_i)^2$ where W_i are the layer weight matrices as described in [67]. For each regularization rate in $\alpha \in [0, 0.01, 0.025, 0.05, 0.075, 0.1]$, we trained 3 different times with a different random seed.

C.8 Geometric complexity in the presence of Adam

We replicate the implicit and explicit regularization experiment using Adam, which is widely used in practice. In this case the conclusions are less clear than with vanilla SGD or SGD with momentum. While higher learning rates, and higher explicit flatness, L2, and spectral regularization still put a regularizing pressure on the geometric complexity for most of the training, the effect of batch size on geometric complexity is not clear. This may be that the local built-in re-scaling of the gradient sizes in Adam affects the pressure on the geometric complexity in complex ways when the batch size changes.

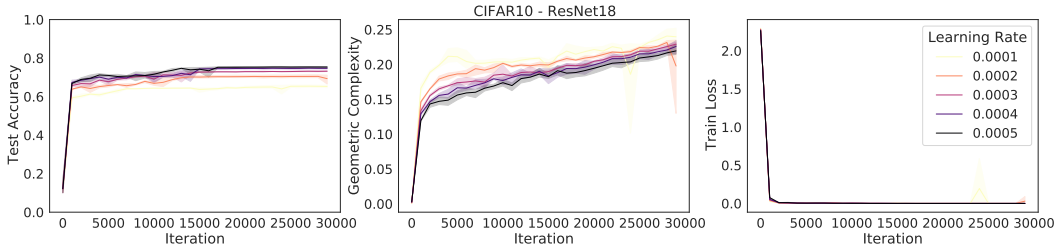

Figure 31: **Geometric complexity decreases with higher learning rates on CIFAR10 trained using Adam with** $b1 = 0.9$**,** $b2 = 0.999$: We trained a selection of ResNet18 with batch size of 512 for 30000 steps using Adam with a momentum of 0.9. For each learning rate $\alpha \in [0.0001, 0.0002, 0.0003, 0.0004, 0.0005]$, we trained 3 different times with a different random seed.

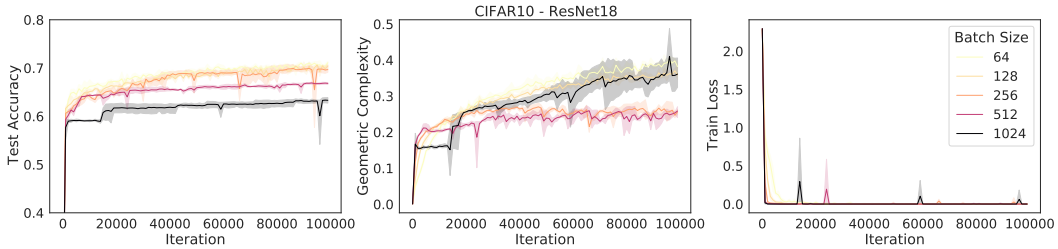

Figure 32: **The relation between geometric complexity and batch size is ambiguous on CIFAR10 trained with Adam**: We trained a selection of ResNet18 with learning rate of 0.0001 for 100000 steps using using Adam with $b1 = 0.9$, $b2 = 0.999$. For each batch size in $\alpha \in [64, 128, 256, 512, 1024]$, we trained 3 different times with a different random seed.

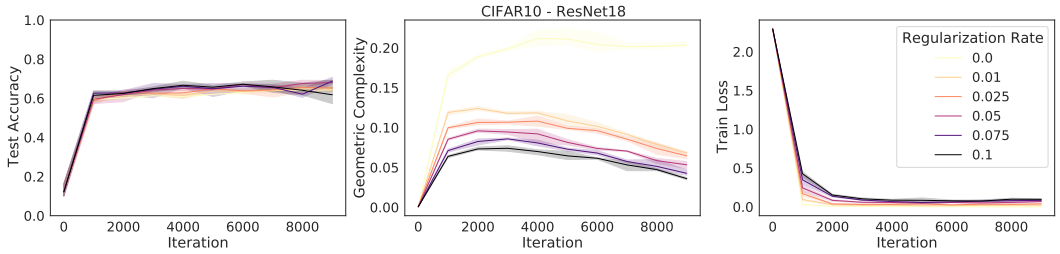

Figure 33: **Geometric complexity decreases with increased L2 regularization on CIFAR10 trained with Adam**: We trained a selection of ResNet18 with learning rate of 0.0001 with batch size of 512 for 10000 steps using using Adam with $b1 = 0.9$, $b2 = 0.999$. For each regularization rate in $\alpha \in [0, 0.01, 0.025, 0.05, 0.075, 0.1]$, we trained 3 different times with a different random seed.

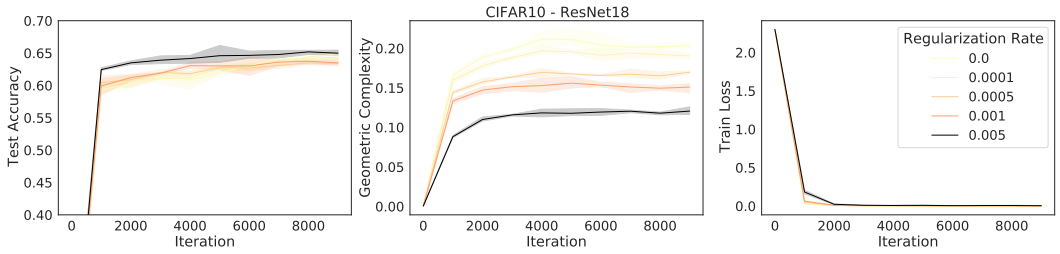

Figure 34: **Geometric complexity decreases with increased flatness regularization on CIFAR10 trained with Adam**: We trained a selection of ResNet18 with learning rate of 0.0001 with batch size of 512 for 10000 steps using using Adam with $b1 = 0.9$, $b2 = 0.999$. We regularized the loss by adding to it the gradient penalty $\alpha\|\nabla_\theta L_B(\theta)\|^2$ where L_B is the batch loss. For each regularization rate in $\alpha \in [0, 0.0001, 0.0005, 0.001, 0.005]$, we trained 3 different times with a different random seed.

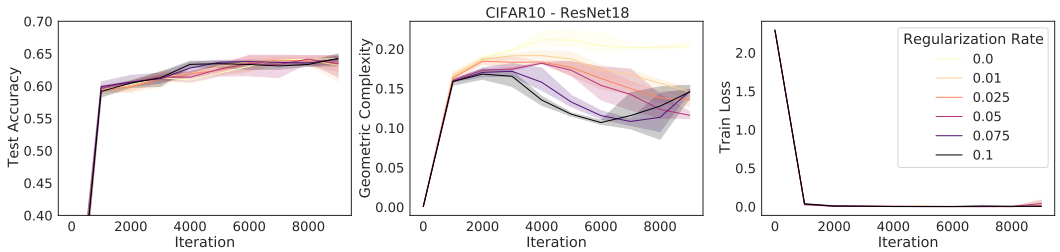

Figure 35: **Geometric complexity decreases with increased spectral regularization on CIFAR10 trained with Adam**: We trained a selection of ResNet18 with learning rate of 0.0001 with batch size of 512 for 10000 steps using Adam with $b1 = 0.9$, $b2 = 0.999$. We regularized the loss by adding to it the spectral norm penalty $\alpha/2 \sum_i \sigma_{\max}(W_i)^2$ where W_i are the layer weight matrices as described in [67]. For each regularization rate in $\alpha \in [0, 0.01, 0.025, 0.05, 0.075, 0.1]$, we trained 3 different times with a different random seed.

D Comparison of the Geometric Complexity to other complexity measures

One of the primary challenges in deep learning is to better understand mechanisms or techniques that correlate well with (or can imply a bound on) the generalization error for large classes of models. The standard approach of splitting the data into a train, validation and test set has become the de facto way to achieve such a bound. With this goal in mind, a number of complexity measures have been proposed in the literature with varying degrees of theoretical justification and/or empirical success. In this section we compare our geometric complexity measure with other, more familiar complexity measures such as the Rademacher complexity, VC dimension and sharpness-based measures.

D.1 Rademacher Complexity

Perhaps the most historically popular and widely known complexity measure is the Rademacher complexity [7, 9, 51, 52]. Loosely speaking the Rademacher complexity measures the degree to which a class of functions $\mathcal{H}$ can fit random noise. The idea behind this complexity measure is that a more complex function space is able to generate more complex representation vectors and thus, on average, produce learned functions that are better able to correlate with random noise than a less complex function space.

To make this definition more precise and frame it in the context of machine learning (see also [68]), given an input feature space X and a target space Y, let $\mathcal{G}$ denote a family of loss functions $L : \mathcal{Z} = X \times Y \to \mathbb{R}$ associated with a function class $\mathcal{H}$. Notationally,

$$\mathcal{G} = \{g : (x, y) \mapsto L(h(x), y) : h \in \mathcal{H}\}.$$

We define the *empirical Rademacher complexity* as follows:

Definition D.1 (Empirical Rademacher Complexity)**.** *With $\mathcal{G}$ as above, let $S = \{z_1, z_2, \ldots, z_m\}$ be a fixed sample of size m of elements of $\mathcal{Z} = X \times Y$. The* empirical Rademacher complexity of $\mathcal{G}$ *with respect to the sample S is defined as:*

$$\widehat{\mathfrak{R}}_S(\mathcal{G}) = \mathbb{E}_{\boldsymbol{\sigma}}\left[\sup_{g\in\mathcal{G}} \frac{1}{m}\sum_{i=1}^{m} \sigma_i g(z_i)\right],$$

where $\boldsymbol{\sigma} = (\sigma_1, \sigma_2, \ldots, \sigma_m)^\intercal$, with the σ_i's being independent uniform random variables which take values in $\{-1, +1\}$. These random variables σ_i are called Rademacher variables.

If we let g_S denote the vector of values taken by function g over the sample S, then the Rademacher complexity, in essence, measures the expected value of the supremum of the correlation of g_S with a vector of random noise $\boldsymbol{\sigma}$; i.e., the empirical Rademacher complexity measures on average how well the function class $\mathcal{G}$ correlates with random noise on the set S. More complex families $\mathcal{G}$ can generate more vectors g_S and thus better correlate with random noise on average, see [68] for more details.

Note that the empirical Rademacher complexity depends on the sample S. The Rademacher complexity is then an average of this empirical measure over the distribution from which all samples are drawn:

Definition D.2 (Rademacher Complexity)**.** *Let $\mathcal{D}$ denote the distribution from which all samples S are drawn. For any integer $m \geq 1$, the* Rademacher complexity of $\mathcal{G}$ *is the expectation of the empirical Rademacher complexity over all samples of size m drawn according to $\mathcal{D}$:*

$$\mathfrak{R}_m(\mathcal{G}) = \mathbb{E}_{S\sim\mathcal{D}^m}[\widehat{\mathfrak{R}}_S(\mathcal{G})].$$

The Rademacher complexity is distribution dependent and defined for any class of real-valued functions. However, computing it can be intractable for modern day machine learning models. Similar to the empirical Rademacher complexity, the geometric complexity is also computed over a sample of points, in this case the training dataset, and is well-defined for any class of differentiable functions. In contrast, the Rademacher complexity (and the VC dimension which we discuss below) measures the complexity for an entire hypothesis space, while the geometric complexity focuses only on single functions. Furthermore, since the Geometric Complexity relies only on first derivatives of the learned model function making it much easier to compute.

D.2 VC dimension

The Vapnik–Chervonenkis (VC) dimension [14, 18, 88] is another common approach to measuring the complexity of a class of functions $\mathcal{H}$ and is often easier to compute than the Rademacher Complexity, see [68] for further discussion on explicit bounds which compare the Rademacher complexity with the VC dimension.

The VC dimension is a purely combinatorial notion and defined using the concept of a *shattering* of a set of points. A set of points is said to be shattered by $\mathcal{H}$ if, no matter how we assign a binary label to each point, there exists a member of $\mathcal{H}$ that can perfectly separate the points; i.e., the growth function for $\mathcal{H}$ is 2^m. The VC dimension of a class $\mathcal{H}$ is the size of the largest set that can be shattered by $\mathcal{H}$.

More formally, we have

Definition D.3 (VC dimension)**.** *Let $\mathcal{H}$ denote a class of functions on X taking values in $\{-1, +1\}$. Define the growth function $\Pi_{\mathcal{H}} : \mathbb{N} \to \mathbb{N}$ as*

$$\Pi_{\mathcal{H}}(m) = \max_{\{x_1,\ldots,x_m\}\subset X} \left|\{(h(x_1), \ldots, h(x_m)) : h \in \mathcal{H}\}\right|.$$

If $\Pi_{\mathcal{H}} = 2^m$ we say $\mathcal{H}$ shatters the set $\{x_1, \ldots, x_m\}$. The VC dimension of $\mathcal{H}$ *is the size of the largest shattered set, i.e.*

$$\textit{VCdim}(\mathcal{H}) = \max\{m : \Pi_{\mathcal{H}}(m) = 2^m\}$$

If there is no largest m, we define VCdim$(\mathcal{H}) = \infty$.

The VC dimension is appealing partly because it can be upper bounded for many classes of functions (see for example, [8]). Similar to the Rademacher complexity, the VC dimension is measured on the entire hypothesis space. The Geometric Complexity, in contrast, is instead measured for given

function within the hypothesis space allowing for more direct comparison between elements within the class $\mathcal{H}$. Computing the VC dimension for a given function set $\mathcal{H}$ may not be always convenient since, by definition, it requires computing the growth function $\Pi_{\mathcal{H}}(m)$ for all subsets of order $m \geq 1$; whereas, the Geometric Complexity relies only on first derivatives and is much easier to precisely compute.

D.3 Sharpness and Hessian related measures

Another broad category of generalization measures concerns the concept of "sharpness" of the local minima; for example, see [40, 47, 64]. Such complexity measures aim to quantify the sensitivity of the loss to perturbations in model parameters. Here a flat minimizer is a point in parameter space where the loss varies only slightly in a relatively large neighborhood of the point. Conversely, the variation of the loss function is less controlled in a neighborhood around a sharp minimizer. The idea is that for sharp minimizers the training function is more sensitive to perturbations in the model parameters and thus negatively impacts the model's ability to generalize; see also [1] which argues that flat solutions have low information content.

The sharpness of a minimizer can be characterized by the magnitude of the eigenvalues of the Hessian of the loss function. However, since the Hessian requires two derivatives this can be computationally costly in most deep learning use cases. To overcome this drawback, [47] suggest a metric that explores the change in values of the loss function f within small neighborhoods of points. More precisely, let $\mathcal{C}_\epsilon$ denote a box around an optimal point in the domain of f and let $A \in \mathbb{R}^{n\times p}$ be a matrix whose columns are randomly generated. The constraint $\mathcal{C}_\epsilon$ is then defined as:

$$\mathcal{C}_\epsilon = \{z \in \mathbb{R}^p : -\epsilon(|(A^+x)_i|) \leq z_i \leq \epsilon(|(A^+x)_i| + 1) \quad \forall i \in \{1, 2, \ldots, p\}\}$$

where A^+ denotes the pseudo-inverse of A and ϵ controls the size of the box. Keskar et al. [47] then define "sharpness" by

Definition D.4 (Sharpness)**.** *Given $x \in \mathbb{R}^n$, $\epsilon > 0$ and $A \in \mathbb{R}^{n\times p}$, the* $(\mathcal{C}_\epsilon)$-sharpness of f at x is defined as

$$\phi_{x,f}(\epsilon, A) = \frac{(\max_{y\in\mathcal{C}_\epsilon} f(x + Ay)) - f(x)}{1 + f(x)} \times 100$$

Another related complexity measure is the *effective dimension* which is computed using the spectral decomposition of the loss Hessian [63]. Since the effective dimension relies on the Hessian, it causes flat regions in the loss surface to also be regions of low complexity w.r.t. this measure.

Effective dimensionality [62] was originally proposed to measure the dimensionality of the parameter space determined by the data and is computed using the eigenspectrum of the Hessian of the training loss.

Definition D.5 (Effective dimensionality of a symmetric matrix)**.** *The* effective dimensionality *of a symmeteric matrix $A \in \mathbb{R}^{k\times k}$ is defined as*

$$N_{\textit{eff}}(A, z) = \sum_{i=1}^{k} \frac{\lambda_i}{\lambda_i + z}$$

where λ_i are the eigenvalues of A and $z > 0$ is a regularization constant.

When used in the context of measuring the effective dimension of a neural network $f(x; \theta)$ with inputs x and parameters $\theta \in \mathbb{R}^k$, we take A to be the Hessian of the loss function; i.e., the $k \times k$ matrix of second derivatives of the loss $\mathcal{L}$ over the data distribution $\mathcal{D}$ defined as $\text{Hess}_\theta = -\nabla^2\mathcal{L}(\theta, \mathcal{D})$. Furthermore, the computation of the effective dimension involves both double derivatives (to compute the Hessian) but also evaluation of the eigenvalues of the resulting matrix. This can introduce a prohibitive cost in computation for many deep learning models.

The Geometric Complexity differs from these complexity measures in a meaningful way. Sharpness and effective dimension are ultimately concerned with the behavior and derivatives of the loss function with respect to the parameter space. The Geometric Complexity, however, is measured using derivatives of the learned model function with respect to the model inputs. Furthermore, the Geometric Complexity is computed using only a single derivative, making it computationally tractable to measure and track.

That being said, these sharpness measures and the Geometric Complexity are also quite related. For example, as explained by the Transfer Theorem in Section 5, for neural networks these flat regions are also the regions of low loss gradient and thus of low GC. Furthermore, similar to the GC, the effective dimension can also capture the double descent phenomena and, in [63], the authors argue the effective dimension provides an efficient mechanism for model selection.