GrokAlign: Geometric Characterisation and Acceleration of Grokking

Thomas Walker[1]* **Ahmed Imtiaz Humayun**[2]† **Randall Balestriero**[3]

Richard Baraniuk[1]

[1]Department of Electrical and Computer Engineering, Rice University
[2]Google Research
[3]Department of Computer Science, Brown University

Abstract

A key challenge for the machine learning community is to understand and accelerate the training dynamics of deep networks that lead to delayed generalisation and emergent robustness to input perturbations, also known as grokking. Prior work has associated phenomena like delayed generalisation with the transition of a deep network from a linear to a feature learning regime, and emergent robustness with changes to the network's functional geometry, in particular the arrangement of the so-called linear regions in deep networks employing continuous piecewise affine nonlinearities. Here, we explain how grokking is realised in the Jacobian of a deep network and demonstrate that aligning a network's Jacobians with the training data (in the sense of cosine similarity) ensures grokking under a low-rank Jacobian assumption. Our results provide a strong theoretical motivation for the use of Jacobian regularisation in optimizing deep networks – a method we introduce as GrokAlign – which we show empirically to induce grokking much sooner than more conventional regularizers like weight decay. Moreover, we introduce centroid alignment as a tractable and interpretable simplification of Jacobian alignment that effectively identifies and tracks the stages of deep network training dynamics. Accompanying webpage and code.

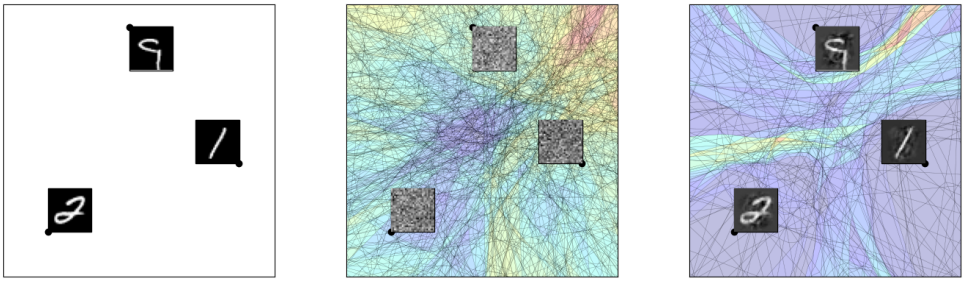
1 Introduction

Deep networks are known to have emergent properties during prolonged training that are essential to understand to facilitate their reliable and effective training. *Delayed generalisation* involves the test accuracy increasing long after train accuracy has saturated, a phenomenon initially termed *grokking* [4]. Delayed generalisation spans multiple deep architectures and domains, including transformers on algorithmic tasks [4] and natural language processing [5], and fully connected networks performing image-classification [6]. Subsequently, the grokking concept has been expanded to include *delayed robustness* [7], which involves prolonged training inducing a robustification of the deep network to input perturbations. Ideally we would accelerate the onset of both generalisation and robustness in deep networks, though in practice there has been an observed tension between them [8, 9].

Despite a high-level understanding of grokking, there exists no foundational explanation for why it occurs nor a practical and interpretable framework for accelerating a deep network's training dynamics to reach the grokked state more efficiently.

*Correspondence to tw78@rice.edu
†Work done before joining Google.

Preprint.



Data Points | Centroids at Memorisation | Centroids at Generalisation

Figure 1: For a deep network to grok, its Jacobians should *align* such that the sum of their rows are cosine-similar to the point at which they were computed; we dub this condition *centroid aligned*. We train a ReLU network on the MNIST dataset [1] using GrokAlign. We take three training data points, **left**, and observe the linear regions (using SplineCam [2]) of the deep network along with the centroids [3] of the three data points when it has memorized the training data, **centre**, and when it has generalised, **right**. We colour the linear regions according to the norm of the linear operator acting upon them. (Additional figures can be found in Figure 11).

Prior work in this space attributes delayed generalisation to the transition of a deep network from a linear to a feature learning regime [10–12]. Studies have explored this dynamic from the perspective of the neural tangent kernel [12, 13], the adaptive kernel [11, 14], and mechanistic interpretability [15, 16]. On the one hand, these works have arrived at an array of sufficient conditions for inducing generalisation, including weight-norm at initialisation [6], weight-decay [10, 15], dataset size [16], and output or label scaling [12]. On the other hand, delayed generalisation and robustness has been attributed to the evolution of the functional geometry of the network [7], which is a reference to the arrangement of the so-called linear regions of a continuous piecewise affine network. The Jacobian matrices of a deep network have also been identified as intimately related to their robustness [17–21].

In this paper, we prove theoretically and demonstrate empirically that Jacobian norm constraints induces grokking in deep networks. Moreover, we develop a tractable and interpretable approach for monitoring and accelerating grokking in practice based on an efficient summarization of the Jacobian called the centroid.

This paper makes three main contributions. First, we demonstrate that deep networks that have optimized their loss function have *aligned* Jacobians at the training data points, in the sense that the rows of the Jacobians at the training points are simply scalar multiples of those points. Deep networks with aligned Jacobians have been empirically demonstrated to be robust [20, 21], and we prove rigorously that they are optimally robust amongst all rank-one Jacobians. Since deep network training dynamics tend to bias the Jacobian towards a low-rank matrix [22–26], we conclude that the *the cause of grokking is the alignment of the deep network's Jacobian matrices*. Through this theory we introduce *GrokAlign*, which is the use of Jacobian regularisation to ensure and accelerate grokking.

Second, since working with Jacobian matrices in practice is computationally expensive, and current strategies to align the Jacobians are cumbersome [21], we propose to summarize the Jacobian matrices via the sum of their rows. This vector can be efficiently computed through Jacobian vector products [27], and it has a strong geometrical interpretation in the spline theory of deep learning [28], where it corresponds to the *centroid* of the linear region containing the data point of interest. Theoretically, the centroids are connected to the neural tangent kernel, and empirically they offer efficiently computable metrics for monitoring and detecting the emergence of grokking.

Third, we explore the practical significance of Jacobian and centroid alignment as a framework through which to consider the dynamics of deep network training. We demonstrate that centroid alignment can reliably identify key phases in the training dynamics of deep networks and when additional training could be beneficial. Furthermore, we illustrate the effectiveness of GrokAlign as a strategy for controlling the training dynamics of deep networks that lead to grokking.

In summary, in Section 2, we theoretically identify Jacobian alignment as the grokked state of a deep network which can be arrived at by regularising the Jacobian matrices of the deep network during training – a method we introduce as GrokAlign. In Section 3, we demonstrate how we can

equivalently track Jacobian alignment through centroid alignment. In Section 4, we put this into practice by demonstrating that we can use centroid alignment to identify the key phases of deep network training and we can use GrokAlign to induce robustness, inhibit or accelerate grokking and control the learned solutions of deep networks.

2 Jacobian Regularisation Explains Grokking

Let $f : \mathbb{R}^d \to \mathbb{R}^C$ be a deep network. Here we focus on the classification setting so that the prediction of the deep network at a point $\mathbf{x}$ is taken to be $\operatorname{argmax}(f(\mathbf{x}))$. In this setting, the deep network is trained on a data set $\{(\mathbf{x}_p, y_p)\}_{p=1}^m$ – where $\mathbf{x}_p \in \mathbb{R}^d$ and $y_p \in \mathbb{R}$ is its corresponding class – under some loss function $\mathcal{L} = \frac{1}{m}\sum_{p=1}^m \ell\left(f\left(\mathbf{x}_p\right), y_p\right)$. Let $J_{\mathbf{x}}(f)$ be the Jacobian of f at $\mathbf{x}$.

Definition 1. *A deep network is **Jacobian-aligned** at $\mathbf{x} \in \mathbb{R}^d$ if $J_{\mathbf{x}}(f) = \mathbf{c}\mathbf{x}^\top$ for some vector $\mathbf{c} \in \mathbb{R}^d$.*

A deep network is said to have *generalised* when it learns to extrapolate beyond the training set and perform well on unseen inputs, and it is said to be *robust* when applying perturbations to inputs does not change the behaviour of the deep network drastically. The emergence of generalisation is typically formalised under the feature learning regime of training [10–12], whilst the robustness of deep networks has been connected to its Jacobians [20, 21]. The delayed onset of both these properties is encapsulated in the grokking phenomenon [4, 7].

Let us suppose that f is a continuous piecewise affine deep network – which includes a broad class of architectures, including ReLU feedforward networks, recurrent networks, convolutional neural networks, and residual networks with piecewise linear activation functions [28]. Such a deep network has a representation of the form $f(\mathbf{x}) = A_{\omega_{\mathbf{x}}}\mathbf{x} + B_{\omega_{\mathbf{x}}}$ where $A_{\omega_{\mathbf{x}}} \in \mathbb{R}^{C\times d}$ and $B_{\omega_{\mathbf{x}}} \in \mathbb{R}^C$. More specifically, $A_{\omega_{\mathbf{x}}}$ and $B_{\omega_{\mathbf{x}}}$ are the parameters for the affine transformation operating on the linear region $\omega_{\mathbf{x}}$ encompassing $\mathbf{x}$. The *functional geometry* of f is the disjoint union of these linear regions, which is a finite-partition of the input space into a collection of convex polytopes [3]. Note that in this setting $J_{\mathbf{x}}(f) = A_{\omega_{\mathbf{x}}}$.

Theorem 2. *Let $\mathcal{L}$ be the cross-entropy or mean-squared error loss function. Then the continuous piecewise affine deep network f minimising $\mathcal{L}$ under the constraints that $\left\|J_{\mathbf{x}_p}(f)\right\|_F^2 \leq \alpha$ and $B_{\omega_{\mathbf{x}_p}} = \mathbf{0}$ for every $p = 1, \ldots, m$, is Jacobian-aligned. (Proof in Appendix E).*

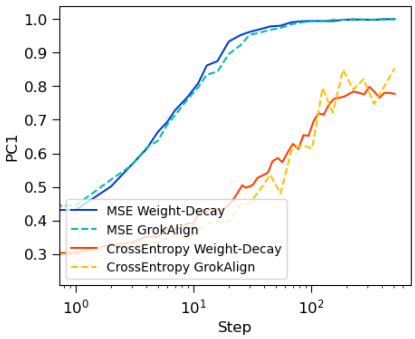


Figure 2: Under weight-decay and GrokAlign, the effective rank of the Jacobian matrices evaluated at the training data tends towards rank one. Here we trained ReLU networks on the MNIST classification task [1] under the mean-squared error and cross-entropy loss functions using the AdamW optimizer [29]. Throughout training, we recorded the average explained variance of the first principal component of the Jacobians evaluated at the training data (PC1), namely $\frac{\sigma_1^2}{\sum_{i=1}^r \sigma_i^2}$ where σ are the singular values of the Jacobian. When this normalized value equals one, the Jacobian matrix is rank one.

Theorem 2 demonstrates that Jacobian-aligned deep networks are optimal in the sense of optimising the training objective. Combined with prior works [20, 21], we can also infer that Jacobian-aligned deep networks are robust.[3] We support this with the following.

Theorem 3. *If $A_{\omega_{\mathbf{x}}}$ is a rank-one matrix and $B_{\omega_{\mathbf{x}}} = \mathbf{0}$, then the local mapping on the linear region $\omega_{\mathbf{x}}$ is optimally robust with respect to ℓ_2 perturbations when $A_{\omega_{\mathbf{x}}} = \mathbf{c}\mathbf{x}^\top$, where the maximum entry of $\mathbf{c}$ is at the index of the class of $\mathbf{x}$. (Proof in Appendix E).*

In practice, the dynamics of deep network training biases toward low rank weight matrices [22–26], and thus low rank Jacobians (see Figure 2). Hence, from Theorem 3, we determine that delayed robustness ought to necessarily involve the Jacobian alignment of deep networks.

[3] Although, in these prior works, the notion of alignment considers only the row of the Jacobian matrix corresponding to the class of the input.

By explicitly enforcing the Jacobian norm constraint of Theorem 2 – a method we introduce as GrokAlign – we can guarantee that optimising the training objective will lead to a grokked network. More specifically, GrokAlign involves appending the average Frobenius norm[4] of the Jacobian matrices at the training data to the loss function with some weighting coefficient, λ_{Jac}.

Reassuringly, this form of Jacobian regularisation has been demonstrated to improve the robustness of deep networks in prior work [17–19, 30]. Moreover, forcing the network to maintain a low Lipschitz constant, which is equivalent to having a low Jacobian norm, has been identified to balance the observed trade-off between generalisation and robustness [31].

Weight-decay could also be a viable strategy of enforcing the constraint of Theorem 2; however, it is less direct. Indeed, in some cases weight-decay has proven effective for inducing grokking [4, 15, 16, 32], while in other cases it has shown to be insufficient for grokking [12].

In summary, we have identified that grokking requires the alignment of a deep network's Jacobian which, due to Theorem 2, only arises during training if the deep network's Jacobians are regularised to remain bounded. Thus we proposed GrokAlign as a method for doing this.

3 The Centroid Alignment Perspective

Centroids. Recall that the functional geometry of a continuous piecewise affine deep network refers to the arrangement of its linear regions. That is, the disjoint union of $\{\omega_{\mathbf{x}}\}_{\mathbf{x}\in\mathbb{R}^d/\sim}$ where $\sim$ denotes the equivalence class $\mathbf{x}_1 \sim \mathbf{x}_2$ if and only if $\mathbf{x}_2 \in \omega_{\mathbf{x}_1}$ and vice-versa. Of importance is the fact that the functional geometry can be parametrised with a collection of parameters $\{(\mu_{\mathbf{x}}, \tau_{\mathbf{x}})\}_{\mathbf{x}\in\mathbb{R}^d/\sim} \subseteq \mathbb{R}^d \times \mathbb{R}$, termed the *centroids* and *radii*, according to a power diagram subdivision [3].

Theorem 4. *For a continuous piecewise affine deep network, $\mu_{\mathbf{x}} = (J_{\mathbf{x}}(f))^\top \mathbf{1}$. (Proof in Appendix E).*

Theorem 4 tells us that centroids provide a summarisation mechanism for the Jacobian matrices of any deep network in way that has an elegant geometrical interpretation when the network is continuous piecewise affine. Importantly, the centroid can be computed through a Jacobian vector product, which is more computationally efficient than directly computing the Jacobian [27].

Definition 5. *A deep network is **centroid-aligned** at $\mathbf{x} \in \mathbb{R}^d$ if $\mu_{\mathbf{x}} = c\mathbf{x}$ for some constant $c \in \mathbb{R}$.*

The geometrical consequences of centroid alignment can be visualised vividly in Figure 1. An aligned centroid can be linked to the *region migration* phenomenon observed in Humayun et al. [7], which was used as an explanation for delayed robustness. Region migration describes the process of a deep network migrating its linear regions from the data points to the decision boundary. Therefore, we can already observe that considering centroid alignment will be useful from the perspective of trying to understand the dynamics of deep network training.

Proposition 6. *A Jacobian-aligned deep network is centroid-aligned. (Proof in Appendix E).*

From Proposition 6 it follows that we can consider centroid alignment as an alternative to Jacobian alignment. Although centroid alignment is a weaker property than Jacobian alignment, we will demonstrate that it has explicit connections to feature learning through the neural tangent kernel.

Neural Tangent Kernel. Suppose a deep network has parameters θ. Then the neural tangent kernel [13] between $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^d$ is taken to be $\Theta(\mathbf{x}, \mathbf{x}') = \nabla_\theta f_\theta(\mathbf{x}) (\nabla_\theta f_\theta(\mathbf{x}'))^\top$. The *linear* and *feature* learning regimes of deep network training are characterised by having relatively constant or dynamic neural tangent kernels, respectively [33–35]. The former identifies when the deep network approximates a linear function, whereas the latter involves the deep network's nonlinearities.[5]

The Dynamics of Centroids. For simplicity we will explore the centroid dynamics of a two-layer network of the form $f_\theta(\mathbf{x}) = W^{(2)}\left(\sigma\left(W^{(1)}\mathbf{x}\right)\right)$, where $W^{(2)} \in \mathbb{R}^{d^{(2)}\times d^{(1)}}$, $W^{(1)} \in \mathbb{R}^{d^{(1)}\times d}$, and σ is a piecewise affine nonlinearity (e.g. ReLU). We omit bias terms to ensure the zero bias assumption

[4]The Frobenius norm of a matrix is equal to the square root of the sum of the squares of its components.

[5]Lazy and rich are also commonly used terms to refer to these two regimes.

of Theorem 2. We will suppose the deep network is being trained using full-batch gradient descent with a learning rate of η.

Lemma 7. *In the setting outlined above, we have* $\mu_{\mathbf{x}} = \left(W^{(2)} Q_{\mathbf{x}} W^{(1)}\right)^{\top} \mathbf{1}$*, where* $Q_{\mathbf{x}} := \operatorname{diag}\left(\sigma'\left(W^{(1)}\mathbf{x}\right)\right)$*. (Proof in Appendix E).*

To make the connection to feature learning explicit, we will consider the deep network to have a scalar output. However, we provide a treatment of vector-output deep networks in Appendix A, where we make an analogous connection between centroids dynamics and feature learning. In the scalar-output setting we suppose that the deep network is being trained with the cross-entropy loss function.

Lemma 8. *In the setting described above, with* $d^{(2)} = 1$*, the neural tangent kernel of* f_θ *between* $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^d$ *is given by*

$$\Theta\left(\mathbf{x}, \mathbf{x}'\right) = \sigma\left(W^{(1)}\mathbf{x}\right)^{\top} \sigma\left(W^{(1)}\mathbf{x}'\right) + \left(\mathbf{x}^{\top}\mathbf{x}'\right)\left(W^{(2)} Q_{\mathbf{x}} Q_{\mathbf{x}'}\left(W^{(2)}\right)^{\top}\right).$$

(Proof in Appendix E).

Theorem 9. *In the setting of Lemma 8, we have* $\partial_t\left(\langle \mathbf{x}, \mu_{\mathbf{x}} \rangle\right) = \frac{\eta}{m} \sum_{p=1}^{m} \Theta\left(\mathbf{x}, \mathbf{x}_p\right) m_{\mathbf{x}_p}$*, where* $m_{\mathbf{x}_p} = y_p - \frac{1}{1+\exp(-f_\theta(\mathbf{x}_p))}$*. (Proof in Appendix E).*

Theorem 9 says that the inner product between some point in the input space, $\mathbf{x}$, and its corresponding centroid, $\mu_{\mathbf{x}}$, is a weighted sum of the neural tangent kernel of the point with the points in the training data. In particular, a changing inner product implies a dynamic neural tangent kernel, which corresponds to the feature learning regime of training.

More specifically, if the inner product $\langle \mathbf{x}, \mu_{\mathbf{x}} \rangle$ changes by δ, then the alignment will change by $\frac{\delta}{\|\mathbf{x}\|\|\mu_{\mathbf{x}}\|}$. When optimizing a deep network with GrokAlign, we would expect the centroid norm to be low as the centroid is equal to the sum of the rows of the Jacobian (see Theorem 4). Thus, the feature learning regime will be identified by centroid alignment.

Consequently, since Jacobian-aligned deep networks are centroid-aligned and centroid alignment is an indicator of feature learning, we have determined that we can use the degree of centroid alignment as a metric for effectively monitoring the emergence of generalisation and robustness in deep network training dynamics. We will now explore this further via a range of numerical experiments.

4 Experiments

To compute the centroid alignment of a deep network, we compute the centroid for an input point using Theorem 4 and then compute the cosine similarity between this and the input point. Likewise, we can obtain the centroid inner product. Due to the zero bias assumption of Theorem 2 and Theorem 3, we will, unless stated otherwise, omit bias terms in the deep networks we consider.

4.1 Centroid Alignment Identifies the Feature Learning Regime

First, we verify Theorem 9 and the conclusions stemming from it in an MNIST [1] two-class classification setting using a two-layer scalar-output ReLU network. Figure 3 demonstrates that the inner product between a point and its linear region's centroid changes in accordance with the neural tangent kernel, meaning that centroid alignment identifies the feature learning regime.

Due to the fact that the norms of the centroids increase during training, eventually the increasing centroid inner product experienced during the feature learning regime no longer translates to centroid alignment. This supports the observation that standard training techniques do not maintain a bounded Jacobian norm [31], which highlights the necessity of a method such as GrokAlign to ensure the realisation of a Jacobian-aligned deep network in practice. Without using a method like GrokAlign during training, the initialisation of the network significantly influences its subsequent dynamics. In Appendix B we use centroid alignment to explore this.

4.2 Centroid Alignment Identifies Delayed Robustness

Having demonstrated that centroid alignment identifies the feature learning regime, we now demonstrate that it can be used to identify the onset of robustness.

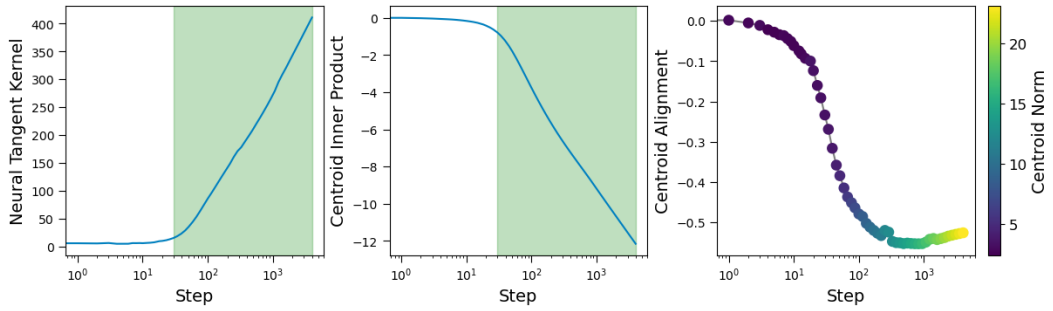


Figure 3: Theorem 9 holds in practice: we observe that a changing inner product indeed corresponds to a feature learning regime. Here we train a two-layer scalar-output ReLU network using the binary-cross-entropy loss function to distinguish between the zero and one class of the MNIST dataset [1]. We train the model using full-batch gradient descent for 4000 steps at a learning rate of 0.01. At the beginning of training we fix a point from the training set and compute the average value of the neural tangent kernel between itself and the other points from the training set, **left**. We then compute the centroid of the point using Theorem 4 to then obtain the inner product, **centre**, and its alignment, **right**. In the right plot we record the norm of the centroid and encode it in the colours of the markers. In the **left** and **centre** plots, we identify the feature learning regime using a green shaded area, determined by the neural tangent kernel changing value.

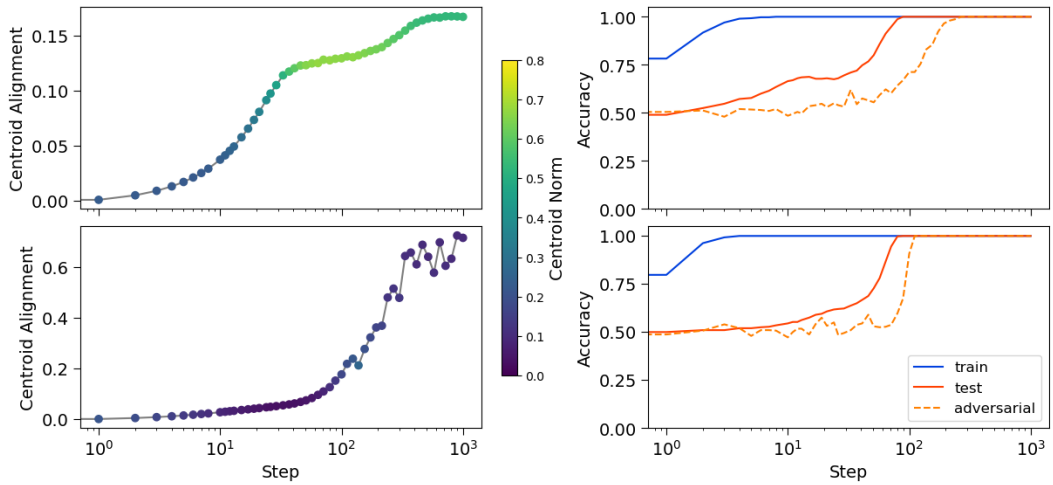


Figure 4: Centroid alignment of a point from the training set identifies the generalisation and robustness of a deep network. We study a two-layer network of width 2048 learning the XOR classification task described in Section 4.2. In the **top** row we train the network using full-batch gradient descent under the mean-squared error loss function with a learning rate of 0.1, weight-decay of 0.1 and for 1000 steps. We monitor both the alignment of a training point to its centroid as well as the robustness of the network. To measure robustness, we take the entries from the test set and apply Gaussian perturbations of varying standard deviations to the last 39998 components. In the **bottom** row we additionally apply GrokAlign with λ_{Jac} equal to 1.0. In the **left** plots we are illustrating the centroid norms with the colours of the markers.

We adopt a set-up similar to that of Xu et al. [36] entailing a scalar-output two-layer fully connected network grokking on XOR cluster data. Note that this network trivially has rank-one Jacobians at every point in the input space. The XOR cluster data contains 40000-dimensional vectors of the form $\mathbf{x} = \left(x_1, x_2, \tilde{\mathbf{x}}^\top\right)^\top \in \mathbb{R}^{40000}$, where $x_1, x_2 \in \{\pm 1\}$ and $\tilde{\mathbf{x}} \in \mathbb{R}^{39998}$. The 400 samples used to train the network are constructed by sampling entries x_1, x_2 uniformly from $\{\pm 1\}$ and entries of $\tilde{\mathbf{x}}$ uniformly from $\{\pm\epsilon\}$, here we take $\epsilon = 0.05$. The corresponding label of such a sample is $x_1 x_2 \in \{\pm 1\}$. A similar sample is generated as a test set. Therefore, by construction our training data only has *signal* in the first two components, whereas all the other components contain noise. Hence, generalisation would require recognising the pattern of how the first two components lead to the corresponding label, whilst robustness would require the network to not condition its pattern recognition on the last 39998 components.

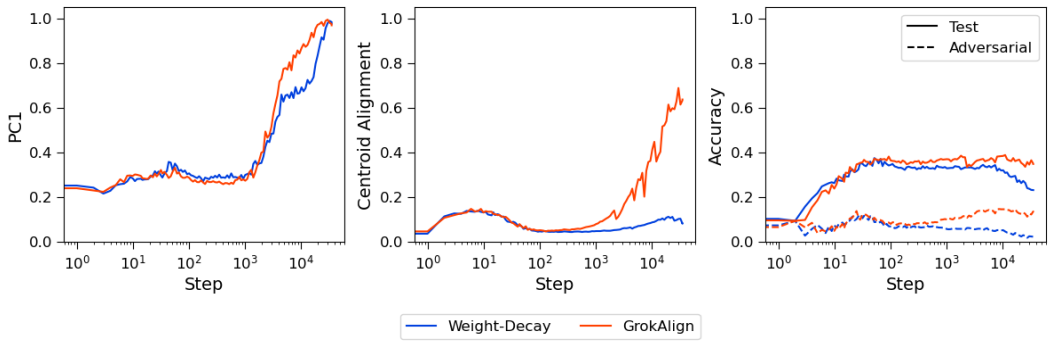


Figure 5: GrokAlign capitalises on the low rank implicit bias of deep network training to induce robustness by aligning the Jacobians of the deep network. Here we train a convolutional neural network with six convolutional layers and two linear layers, with no bias terms, on a 1024 subset of the CIFAR10 dataset [37] under the mean-squared error loss function. We use the AdamW optimizer [29] with a learning of 0.001 and a batch size of 256 to train the network across 36000 steps. In one instance we apply weight-decay at 0.001, and in another instance we apply GrokAlign with λ_{Jac} equal to 0.001. In the **left** plot we compute the average explained variance of the first principal component of the Jacobians as in Figure 2, and in the **centre** plot we record the average centroid alignment on the training set. In the **right** column we record the test accuracy of the model and the accuracy of the model when ℓ_∞ perturbations of amplitude $\frac{4}{255}$ are applied to the test set using Autoattack [38].

Throughout training we track the centroid alignment of a point in the training set to obtain Figure 4. In the top row of Figure 4, we observe that as the network memorises, the centroid alignment does not increase significantly. However, during generalisation, the centroid alignment increases. After a slight plateau in centroid alignment, a further increase correlates with the onset of robustness. In this instance, the rank of the Jacobian of the network at the point under consideration is one, and thus from Theorem 3 robustness can only be achieved through alignment. Critically, we observe the indirectness of weight-decay at imposing the Jacobian norm constraint of Theorem 2. Early on in training, the norms of the centroids increase and eventually inhibit centroid alignment. Since alignment is an optimum of the training objective (recall Theorem 2), it is only at this stage that under weight-decay the network is incentivised to reduce the norms of the Jacobian resulting in the onset of robustness.

In the bottom row of Figure 4, we instead see that by applying GrokAlign we directly mitigate this delay and achieve robustness much sooner.

4.3 GrokAlign's Influence on Grokking

Thus far we have shown that centroid alignment provides a valuable perspective on the training dynamics of a deep network, as it identifies stages of generalisation (see Figure 3) and robustification (see Figure 4). We have also shown that standard deep network training, including weight-decay, cannot maintain a Jacobian norm constraint and thus is limited in its ability to Jacobian-align deep networks.

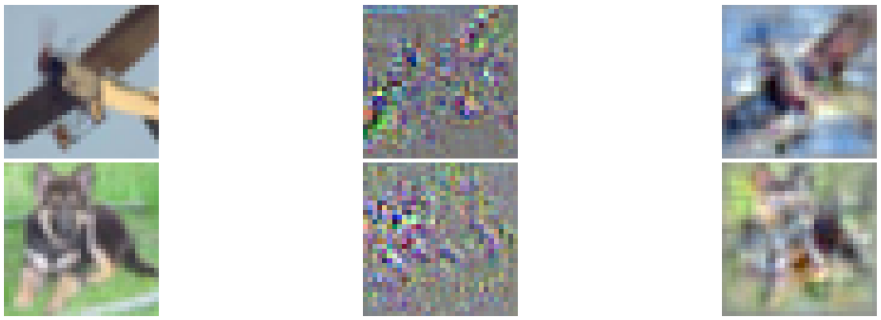
The most direct approach for enforcing the Jacobian norm constraint is through GrokAlign, and here we will explore how this can used to control the deep network's training dynamics that lead to grokking. Recall that GrokAlign regularises the Frobenius norm of the Jacobian matrices of the deep network computed at the training data by appending its value to the loss function. To be computationally efficient, GrokAlign uses an approximation of the Frobenius norm of the Jacobian matrices [19].[6]

Inducing Delayed Robustness. We train convolutional neural networks on the CIFAR10 dataset [37]. From Figure 5 we observe that, by using GrokAlign we can induce Jacobian alignment, as evidenced by the increasing centroid alignment, which translates into improved robustness by capitalising on the diminishing ranks of the Jacobian matrices. With only weight-decay, Jacobian alignment does not manifest, as evidenced by the low centroid alignment, and thus we observe a

[6]We provide code for our implementation of GrokAlign here.

decline in robustness as training progresses. Crucially, by monitoring centroid alignment we can determine that prolonging training is unlikely to improve the properties of the GrokAligned model significantly, since the effective rank of the Jacobians is close to one and the centroid alignments are relatively high and have started plateauing.

Therefore, we have demonstrated that GrokAlign provides a direct strategy for inducing robustness for it aligns the Jacobian's of the deep networks which we can observe by monitoring centroid alignment. Just like Figure 1, we can vividly visualise the consequences of centroid alignment in Figure 6.



Input Centroid Unaligned Centroid Aligned

Figure 6: The geometry of the function of a centroid aligned deep network is more resemblant of the geometry of the training data, indicating that the transformation its applies to local regions of data points is more representative of their features. We compare the two convolutional neural networks from Figure 5 by taking training points, **left**, and computing their centroid. In the **centre** we depict the centroids for the network trained without using GrokAlign, and on the **right** we depict the centroids for the network trained with GrokAlign.

Inhibiting Delayed Generalisation. Using our reasoning, we would expect that if we were to maintain the Jacobian norms at a high-level, then we ought to prevent alignment and thus generalisation. To explore this, we consider a fully connected network and scale up its weights at initialisation to increase the Jacobian norms, much like Liu et al. [6], and apply GrokAlign in different ways.

We observe in Figure 7 that our prediction is correct: minimising the Frobenius norms of the Jacobians leads to generalisation, whilst keeping their value relatively high prevents it. We are able to monitor this by tracking the norms of the centroids, which demonstrates how the centroids provide an effective mechanism to monitor network dynamics.

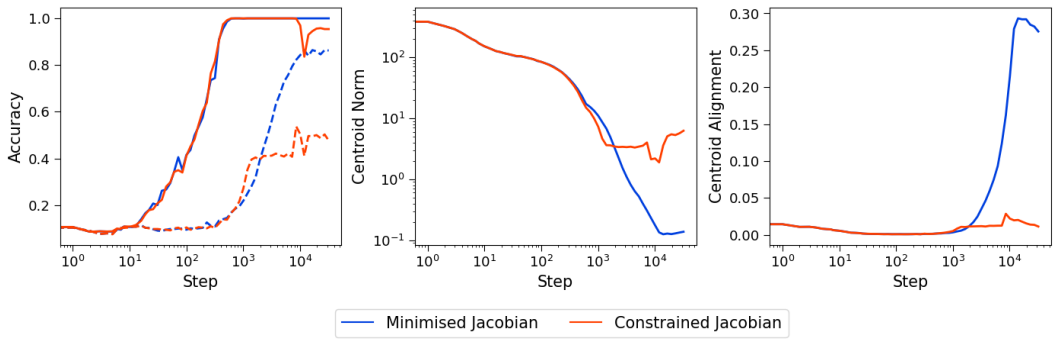


Figure 7: By maintaining the Frobenius norm of the Jacobians at the training data relatively high we can keep the norms of the centroids relatively high which prevents grokking. We take the MNIST grokking set up of Liu et al. [6]. In the minimising case we impose GrokAlign with λ_{Jac} equal to 0.001 during training to minimise the Frobenius norm of the Jacobians at the training data. In the constrained case we apply GrokAlign to maintain the Frobenius norm of the Jacobians computed at the training data at a relatively high level. More specifically, using a regularisation coefficient of 0.001, we append the difference between five and the average Frobenius norm of the training data to the loss function. In the **left** plot we visualise the train and test accuracy with solid and dashed lines respectively. In the **centre** plot we visualise the average norm of the centroids computed at the training data. In the **right** plot we visualise the average centroid alignment.

Accelerating Grokking. Just as we used GrokAlign to inhibit grokking, we can also use it to accelerate grokking. For this we consider the standard MNIST grokking set up of Liu et al. [6] which involves applying weight-decay to a deep network initialised with a large initial weight-norm. From our perspective this results in Jacobians with large norm, inhibiting their alignment.

We compare the effectiveness of GrokAlign to two other known methods of inducing grokking. Grokfast [32] works to improve the rate of grokking by manipulating the gradients during training to amplify certain signals. Tan and Huang [39] motivated adversarial training for accelerating grokking by establishing a connection between robustness and generalisation. The method of adversarial training involves perturbing the inputs during training with noise proportional to the training accuracy of the deep network. In each method we do not manipulate the weight-decay of the training procedure.

From Table 1, we observe that GrokAlign is extremely effective at inducing the grokked state of the deep network in this setting; it arrives at the grokked state in 7.56 times fewer steps and 6.31 times faster than the baseline in the case of the cross-entropy loss function. In contrast, Grokfast provides a relatively lower improvement in the mean-squared error case and is ineffective in the cross-entropy case. Adversarial training does not improve the rate of grokking over the baseline. In particular, adversarial training does not improve the robustness of the model by way of aligning the Jacobian, unlike GrokAlign (see Appendix C).

Table 1: GrokAlign significantly speeds up the rate of grokking. In the **top** table we assume the MNIST set up of Liu et al. [6] with the cross-entropy loss function, and in the **bottom** table we assume it with mean-squared error loss function as the baselines. We repeat the training across ten different random initialisations, where for the cross-entropy loss function we apply GrokAlign with λ_{Jac} equal to 0.001 and with λ_{Jac} equal to 0.0001 for the mean-squared error models. For each run we measure the number of steps and the absolute time, in seconds, taken for the networks to reach 85% test accuracy. In the case of the mean-squared error loss function, we additionally measure the time, in seconds, for the models to go from 20% to 85% test accuracy. We provide the average acceleration (or deceleration) of each method compared to the baseline along with the corresponding standard deviation.

Cross Entropy Loss Function

Regularisation	Number of Steps	Absolute Time (s)
Baseline	–	–
GrokAlign	$\downarrow \mathbf{7.56} \times (\pm 0.82)$	$\downarrow \mathbf{6.41} \times (\pm 0.69)$
Grokfast	$\uparrow 1.01 \times (\pm 0.04)$	$\uparrow 1.03 \times (\pm 0.04)$
Adversarial Training	$\downarrow 1.32 \times (\pm 0.18)$	$\uparrow 1.92 \times (\pm 0.29)$

Mean Squared Error Loss Function

Regularisation	Number of Steps	Absolute Time (s)	Grokking Phase Time (s)
Baseline	–	–	–
GrokAlign	$\downarrow \mathbf{1.69} \times (\pm 0.31)$	$\downarrow \mathbf{1.46} \times (\pm 0.30)$	$\downarrow \mathbf{1.77} \times (\pm 0.53)$
Grokfast	$\downarrow 1.08 \times (\pm 0.17)$	$\downarrow 1.05 \times (\pm 0.23)$	$\downarrow 1.05 \times (\pm 0.27)$
Adversarial Training	$\downarrow 1.23 \times (\pm 0.26)$	$\uparrow 1.92 \times (\pm 0.29)$	$\uparrow 2.02 \times (\pm 0.37)$

Controlling the Learned Solutions of Deep Networks. The computations of centroids is valid without the continuous piecewise affine assumption; it is only their interpretation as characterising a functional geometry that requires the assumption. Therefore, we can examine the centroid alignment of more elaborate models like transformers [40].

Nanda et al. [15] observed that a single layer transformer learning modular addition [4] grokked by learning how to implement an algorithm. An equally viable solution to this problem would be through classification. Since our framework is largely motivated in the classification setting, we can explore the tension between these solutions through centroid alignment.

In Figure 8 we consider the centroid alignment of the transformer model under the standard training pipeline of Nanda et al. [15] with the added utilisation of GrokAlign. As with our previous experiments, we observe that centroid alignment changes in accordance with test accuracy. However, it is

not as salient as in previous cases, and applying GrokAlign does not accelerate the rate of grokking. This is perhaps due to the model learning the algorithmic solution to the task, which is not entirely compatible with the Jacobian and centroid aligned perspective.

We support the idea that GrokAlign biases toward the classification style solution by tracking the Gini coefficient of the embedding and unembedding matrices [15]. A key aspect that allows the transformer to implement its algorithmic solution is the ability to manipulate the embedding and unembedding matrices [15]. In particular, the Gini coefficient of these embedding matrices should be relatively higher when implementing the algorithmic solution. We observe that when applying GrokAlign, these Gini coefficient remain low, indicating that GrokAlign encourages the model to learn the classification style solution which is not the natural solution solution in this particular training regime (see the right plot of Figure 8). Consequently, we observe its ineffectiveness in accelerating grokking.

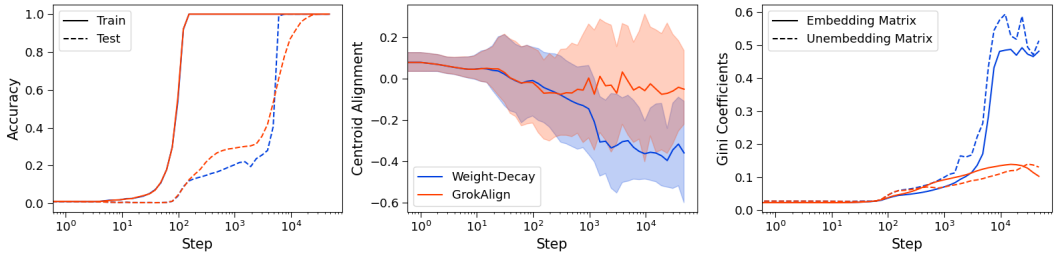


Figure 8: GrokAlign biases a single layer transformer model to learn a *classification* style solution. Here we obtain the centroid alignment statistics for a single layer transformer trained on modular addition [4]. We use the same training pipeline as in Nanda et al. [15], with the added utilisation of GrokAlign with λ_{Jac} equal to 0.001. In the **left** plot we record the accuracies of the transformer model on the train and test set. In the **centre** plot we record the centroid alignment of the transformer model from the continuous embeddings of the discrete input tokens to the output of the model; the shaded region illustrates the maximum and minimum alignment observed with the solid line corresponding to the mean. In the **right** plot we record the Gini coefficients of the embedding and unembedding matrices of the model as proposed in Nanda et al. [15].

If we instead fix the embedding matrix during training, we a priori bias the the model to learn the classification style solution. Under this set up (see Figure 9), we observe that the transformer groks earlier with GrokAlign than with weight-decay.

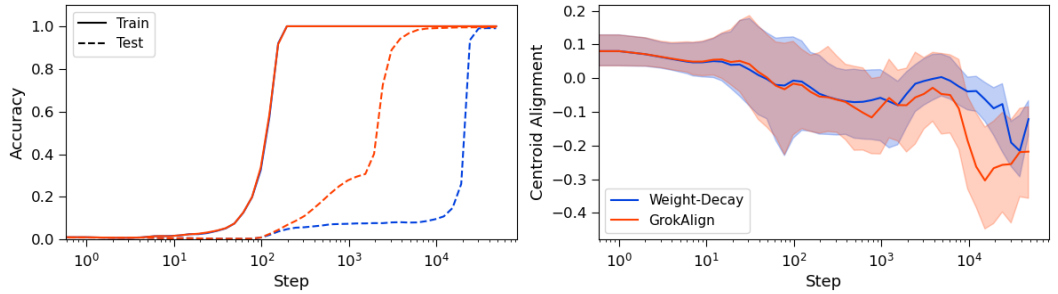


Figure 9: Without the ability to manipulate its embedding matrices, a single layer transformer learns the classification style solution for performing modular addition and thus benefits from being trained with GrokAlign. We adopt the same configurations as in Figure 8, except we fix the embedding and unembedding matrices of the model during training.

5 Discussion

Impacts. We have identified that Jacobian alignment is the cause for grokking, by demonstrating theoretically and empirically that Jacobian-aligned deep networks optimise the loss function under a Jacobian norm constraint and are optimally robust under the low rank bias of training dynamics. Consequently, we identified regularising the norms of the Jacobians of a deep network as an effective strategy for controlling the dynamics of deep networks to instil particular properties – a method we

introduce as GrokAlign. In particular, we showed that using GrokAlign we can induce robustness, inhibit or accelerate grokking and dictate the learned solutions of deep networks.

Since Jacobian matrices are difficult to interpret and costly work with in practice, we constructed the centroid alignment perspective as an alternative strategy to monitor the dynamics of deep networks. This perspective is interpretable due to its relationship with the functional geometry of a deep network and is theoretically meaningful due to its connection to the neural tangent kernel. Using this perspective we were able to identify the onset of generalisation and robustness during deep network training, as well as reason about when prolonging training would improve these important properties.

Limitations. We have developed our theory in the context of continuous piecewise affine networks for convenience, although its implications extend to any deep network architecture, including transformers. Our experiments have mainly considered continuous piecewise networks. Understanding how our perspective holds outside this setting is perhaps warranted. We provide initial results on a transformer in Section 4.3.

Furthermore, some of our reasoning is dependent on the assumption that deep network training dynamics minimise the rank of the weight matrices. Although this has been theoretically demonstrated in particular settings and observed empirically in practice, a deeper characterisation of this phenomenon is necessary.

Acknowledgments and Disclosure of Funding

This work was supported by ONR grant N00014-23-1-2714, ONR MURI N00014-20-1-2787, DOE grant DE-SC0020345, and DOI grant 140D0423C0076.

References

[1] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[2] Ahmed Imtiaz Humayun, Randall Balestriero, Guha Balakrishnan, and Richard Baraniuk. SplineCam: Exact Visualization and Characterization of Deep Network Geometry and Decision Boundaries. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, June 2023.

[3] Randall Balestriero, Romain Cosentino, B. Aazhang, and Richard Baraniuk. The Geometry of Deep Networks: Power Diagram Subdivision. In *Neural Information Processing Systems*, May 2019.

[4] Alethea Power, Yuri Burda, Harri Edwards, Igor Babuschkin, and Vedant Misra. Grokking: Generalization Beyond Overfitting on Small Algorithmic Datasets, January 2022. arXiv:2201.02177.

[5] George Wang, Matthew Farrugia-Roberts, Jesse Hoogland, Liam Carroll, Susan Wei, and Daniel Murfet. Loss landscape geometry reveals stagewise development of transformers. In *High-dimensional Learning Dynamics 2024: The Emergence of Structure and Reasoning*, June 2024.

[6] Ziming Liu, Eric J. Michaud, and Max Tegmark. Omnigrok: Grokking Beyond Algorithmic Data. In *The Eleventh International Conference on Learning Representations*, September 2022.

[7] Ahmed Imtiaz Humayun, Randall Balestriero, and Richard Baraniuk. Deep Networks Always Grok and Here is Why. In *High-dimensional Learning Dynamics 2024: The Emergence of Structure and Reasoning*, June 2024.

[8] Dimitris Tsipras, Shibani Santurkar, Logan Engstrom, Alexander Turner, and Aleksander Madry. Robustness may be at odds with accuracy. In *International Conference on Learning Representations*, 2019.

[9] Hongyang Zhang, Yaodong Yu, Jiantao Jiao, Eric Xing, Laurent El Ghaoui, and Michael Jordan. Theoretically Principled Trade-off between Robustness and Accuracy. In *Proceedings of the 36th International Conference on Machine Learning*, May 2019.

[10] Kaifeng Lyu, Jikai Jin, Zhiyuan Li, Simon Shaolei Du, Jason D. Lee, and Wei Hu. Dichotomy of Early and Late Phase Implicit Biases Can Provably Induce Grokking. In *The Twelfth International Conference on Learning Representations*, January 2024.

[11] Noa Rubin, Inbar Seroussi, and Zohar Ringel. Grokking as a First Order Phase Transition in Two Layer Networks. In *The Twelfth International Conference on Learning Representations*, January 2024.

[12] Tanishq Kumar, Blake Bordelon, Samuel J. Gershman, and Cengiz Pehlevan. Grokking as the transition from lazy to rich training dynamics. In *The Twelfth International Conference on Learning Representations*, January 2024.

[13] Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: convergence and generalization in neural networks. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, 2018.

[14] Inbar Seroussi, Gadi Naveh, and Zohar Ringel. Separation of scales and a thermodynamic description of feature learning in some CNNs. *Nature Communications*, 14(1):908, February 2023.

[15] Neel Nanda, Lawrence Chan, Tom Lieberum, Jess Smith, and Jacob Steinhardt. Progress measures for grokking via mechanistic interpretability. In *The Eleventh International Conference on Learning Representations*, September 2022.

[16] Vikrant Varma, Rohin Shah, Zachary Kenton, János Kramár, and Ramana Kumar. Explaining grokking through circuit efficiency, September 2023. arXiv:2309.02390.

[17] Salah Rifai, Pascal Vincent, Xavier Muller, Xavier Glorot, and Yoshua Bengio. Contractive auto-encoders: explicit invariance during feature extraction. In *Proceedings of the 28th International Conference on Machine Learning*, 2011.

[18] Daniel Jakubovitz and Raja Giryes. Improving DNN Robustness to Adversarial Attacks Using Jacobian Regularization. In *ECCV*, September 2018.

[19] Judy Hoffman, Daniel A. Roberts, and Sho Yaida. Robust Learning with Jacobian Regularization, August 2019. arXiv:1908.02729.

[20] Christian Etmann, Sebastian Lunz, Peter Maass, and Carola Schoenlieb. On the Connection Between Adversarial Robustness and Saliency Map Interpretability. In *Proceedings of the 36th International Conference on Machine Learning*, May 2019.

[21] Alvin Chan, Yi Tay, Yew Soon Ong, and Jie Fu. Jacobian adversarially regularized networks for robustness. In *International Conference on Learning Representations*, 2020.

[22] Thien Le and Stefanie Jegelka. Training invariances and the low-rank phenomenon: beyond linear networks. In *International Conference on Learning Representations*, 2022.

[23] Minyoung Huh, Hossein Mobahi, Richard Zhang, Brian Cheung, Pulkit Agrawal, and Phillip Isola. The low-rank simplicity bias in deep networks. *Transactions on Machine Learning Research*, 2023.

[24] Nadav Timor, Gal Vardi, and Ohad Shamir. Implicit Regularization Towards Rank Minimization in ReLU Networks. In *Proceedings of The 34th International Conference on Algorithmic Learning Theory*, February 2023.

[25] David Yunis, Kumar Kshitij Patel, Samuel Wheeler, Pedro Henrique Pamplona Savarese, Gal Vardi, Jonathan Frankle, Karen Livescu, Michael Maire, and Matthew Walter. Rank minimization, alignment and weight decay in neural networks. In *High-dimensional learning dynamics 2024: The emergence of structure and reasoning*, 2024.

[26] Tomer Galanti, Zachary S. Siegel, Aparna Gupte, and Tomaso A. Poggio. SGD with Weight Decay Secretly Minimizes the Ranks of Your Neural Networks. In *The Second Conference on Parsimony and Learning (Proceedings Track)*, March 2025.

[27] Randall Balestriero and Richard Baraniuk. Fast Jacobian-Vector Product for Deep Networks, April 2021. arXiv:2104.00219.

[28] Randall Balestriero and Richard Baraniuk. A Spline Theory of Deep Learning. In *Proceedings of the 35th International Conference on Machine Learning*. PMLR, July 2018.

[29] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.

[30] S. Gu and Luca Rigazio. Towards Deep Neural Network Architectures Robust to Adversarial Examples. *CoRR*, December 2014.

[31] Yao-Yuan Yang, Cyrus Rashtchian, Hongyang Zhang, Ruslan Salakhutdinov, and Kamalika Chaudhuri. A closer look at accuracy vs. robustness. In *Proceedings of the 34th International Conference on Meural Information Processing Systems*, 2020.

[32] Jaerin Lee, Bong Gyun Kang, Kihoon Kim, and Kyoung Mu Lee. Grokfast: Accelerated Grokking by Amplifying Slow Gradients, June 2024. arXiv:2405.20233.

[33] Lénaïc Chizat, Edouard Oyallon, and Francis Bach. On Lazy Training in Differentiable Programming. In *Advances in Neural Information Processing Systems*, 2019.

[34] Blake Woodworth, Suriya Gunasekar, Jason D. Lee, Edward Moroshko, Pedro Savarese, Itay Golan, Daniel Soudry, and Nathan Srebro. Kernel and rich regimes in overparametrized models. In *Proceedings of Thirty Third Conference on Learning Theory*, July 2020.

[35] Edward Moroshko, Blake Woodworth, Suriya Gunasekar, Jason D. Lee, Nathan Srebro, and Daniel Soudry. Implicit bias in deep linear classification: Initialization scale vs training accuracy. *Advances in Neural Information Processing Systems*, 2020.

[36] Zhiwei Xu, Zhiyu Ni, Yixin Wang, and Wei Hu. Let me grok for you: Accelerating grokking via embedding transfer from a weaker model. In *The Thirteenth International Conference on Learning Representations*, 2025.

[37] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Technical report, University of Toronto / University of Toronto, Toronto, Ontario, 2009.

[38] Francesco Croce and Matthias Hein. Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks. In *Proceedings of the 37th International Conference on Machine Learning*, 2020.

[39] Zhiquan Tan and Weiran Huang. Understanding Grokking Through A Robustness Viewpoint, February 2024. arXiv:2311.06597.

[40] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is All you Need. In *Advances in Neural Information Processing Systems*, 2017.

[41] Jaehoon Lee, Lechao Xiao, Samuel Schoenholz, Yasaman Bahri, Roman Novak, Jascha Sohl-Dickstein, and Jeffrey Pennington. Wide Neural Networks of Any Depth Evolve as Linear Models Under Gradient Descent. In *Advances in Neural Information Processing Systems*, 2019.

[42] Mario Geiger, Stefano Spigler, Arthur Jacot, and Matthieu Wyart. Disentangling feature and lazy training in deep neural networks. *Journal of Statistical Mechanics: Theory and Experiment*, November 2020.

A Centroid Dynamics of Vector-Output Deep Networks

Consider the case of a general vector output, namely $d^{(2)} \geq 2$, with $\mathcal{L}$ being the cross-entropy loss function or the mean-squared error loss function. Namely, we consider

$$\ell\left(f\left(\mathbf{x}_p\right), y_p\right) = -\log\left(\frac{\exp\left(\left[f\left(\mathbf{x}_p\right)\right]_{y_p}\right)}{\sum_{c=1}^{d^{(2)}}\exp\left(\left[f\left(\mathbf{x}_p\right)\right]_c\right)}\right)$$

for the cross-entropy loss function, or

$$\ell\left(f\left(\mathbf{x}_p\right), y_p\right) = \left\|\mathbf{e}_{y_p} - f\left(\mathbf{x}_p\right)\right\|_2^2$$

for the mean-squared error loss function.

Proposition 10. *In the setting described above, we have*

$$\partial_t\left(\langle \mathbf{x}, \mu_{\mathbf{x}}\rangle\right) = \frac{\eta}{m}\sum_{p=1}^{m}\left(\left(\mathbf{m}_{\mathbf{x}_p}^\top W^{(2)} Q_{\mathbf{x}_p} Q_{\mathbf{x}}\left(W^{(2)}\right)^\top \mathbf{1}\right)\langle \mathbf{x}, \mathbf{x}_p\rangle + \mathbf{x}^\top\left(W^{(1)}\right)^\top Q_{\mathbf{x}}\sigma\left(W^{(1)}\mathbf{x}_p\right)\mathbf{m}_{\mathbf{x}_p}^\top \mathbf{1}\right)$$

where

$$\mathbf{m}_{\mathbf{x}_p} = \mathbf{e}_y - \frac{\exp\left(\left[f_\theta\left(\mathbf{x}_p\right)\right]_{y_p}\right)}{\sum_{c=1}^{C}\exp\left(\left[f_\theta\left(\mathbf{x}_p\right)\right]_c\right)}$$

in the case of the cross-entropy loss function and

$$\mathbf{m}_{\mathbf{x}_p} = 2\left(\mathbf{e}_y - f_\theta\left(\mathbf{x}_p\right)\right).$$

Corollary 11. *In the setting of Proposition 10, under the cross entropy loss function, we have*

$$\begin{aligned}\partial_t\left(\langle \mathbf{x}, \mu_{\mathbf{x}}\rangle\right) &= \frac{\eta}{m}\sum_{p=1}^{m}\left(\mathbf{m}_{\mathbf{x}_p}^\top W^{(2)} Q_{\mathbf{x}_p} Q_{\mathbf{x}}\left(W^{(2)}\right)^\top \mathbf{1}\right)\langle \mathbf{x}, \mathbf{x}_p\rangle \\ &:= \frac{\eta}{m}\sum_{p=1}^{m}\frac{\iota_{\mathbf{x},p}}{\|\mathbf{x}_p\|_2}\langle \mathbf{x}, \mathbf{x}_p\rangle\end{aligned}$$

That is, in the context of the cross-entropy loss function, the centroids are aligned in a manner that is proportional to the alignment of their encompassing points with the training data. More specifically, with the intuition that the role of $W^{(2)}$ in f_θ is to be a collection of filters facilitating the classification of each class, the quantity $W^{(2)}Q_{\mathbf{x}_p}\left(W^{(2)}Q_{\boldsymbol{\nu}}\right)^\top \mathbf{1}$ can be thought of as identifying how each feature of $\mathbf{x}_p$ correlates with the features of the region $\omega_{\boldsymbol{\nu}}$. Since $\mathbf{m}\left[\mathbf{x}_p\right]$ is positive on the correct class and negative for the incorrect classes, the term $\iota_{\mathbf{x}',p}$ is largest when $\omega_{\mathbf{x}'}$ has identified features that correlate with the features of $\mathbf{x}_p$ that indicate the class it belongs too. In such a case, the centroid $\mu_{\mathbf{x}'}$ moves in the direction of $\mathbf{x}_p$ to further maximize this correlation. Showing how regions $\omega_{\mathbf{x}'}$ are being optimized to capture the features of classes that help it distinguish it from the other classes. Therefore, we can see neural network training more as a process of allocating linear regions to different features that best distinguishes themselves from the other classes. This would suggest that when we observe the centroids of a layer of a neural network aligning with the data it encompasses, the neural network is performing feature extraction. In particular, the centroid of a training point is incentivised to positively align with itself.

B Initialising for Centroid Alignment

When regularisation is not used to train a deep network, the initialisation of the deep network parameters plays a significant role in determining the extent to which is centroids align. In particular, there is a tension between increasing the rate of change of the centroid inner product and maintaining a low centroid norm to translate the increasing inner product to an increase in alignment.

For example, increasing the rate of change of inner product can be done by increasing the neural tangent kernel, say through scaling the weights or output of the network. However, these will also increase the norm of the centroid. Moreover, such scaling is known to increase the propensity of the network maintaining a linear learning regime [33], along with increasing the width of the neural network [41] and label rescaling [42]. We explore this trade-off by repeating the experiment of Figure 3, but with various scaling of the weights or output of the network.

In Figure 10, we observe that controlling the norm of the centroid is a more effective strategy for translating the feature learning of the deep network into centroid alignment. However, a priori, knowing how to initialise the deep network for favourable alignment dynamics is challenging, hence, in practice some sort of regularisation is necessary.

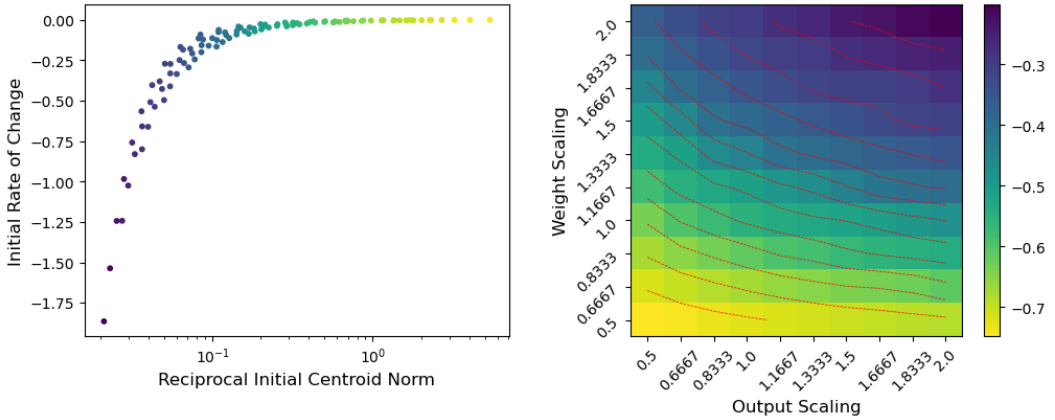


Figure 10: When no regularisation is used, minimising the centroid norm at initialisation is essential for ensuring the alignment of the centroid during training, and output scaling ensures this more effectively than scaling the weights at initialisation. Here we repeat the experiment of Figure 3, but with varying scaling of the weights and output of the network. On the **left** we plot the correlation between the initial rate of change of inner product, as computed by Theorem 9, and the average reciprocal of the norm of the centroids of the training points. We additionally colour the scatter points according to the maximum alignment of the centroid observed during training. On the **right** we observe how the maximum alignment of the centroid observed during training correlates with the different scaling mechanisms.

C Alignment Induced by Adversarial Training

Although both GrokAlign and adversarial training are motivated to induce grokking by improving the model's robustness, the former does this though aligning the functional geometry of the model, whereas the latter does not. We determine this by measuring the cosine similarity between training points and the rows of the Jacobian of the model at those points (see Table 2).

D The Geometry of Centroid Alignment

Centroids have a known connection to the functional geometry of deep networks through the spline theory of deep learning [28]. They form part of the parameter of a region in the power diagram subdivision formulation of deep network's functional geometry [3]. In Figure 1 we demonstrated the implications of centroid alignment on this geometry. More specifically, we showed that it corresponds to the linear regions flattening around inputs of the same class, in a similar way to the region migration phenomenon identified in Humayun et al. [7]. Here we support that claim be illustrating it beyond the input points considered in Figure 1.

Table 2: We compare adversarial training to the baseline and GrokAlign grokking set ups of Table 1 in terms of inducing the alignment of the Jacobian at the training data.

Regularisation	Test Accuracy	Jacobian Row Alignment (max/min)
Baseline	88.9%	−0.254/0.283
GrokAlign	**91.8%**	**−0.509/0.478**
Adversarial Training	88.0%	−0.253/0.274

Regularisation	Test Accuracy	Jacobian Row Alignment (max/min)
Baseline	86.8%	−0.170/0.272
GrokAlign	**88.2%**	**−0.638/0.709**
Adversarial Training	87.8%	−0.263/0.376

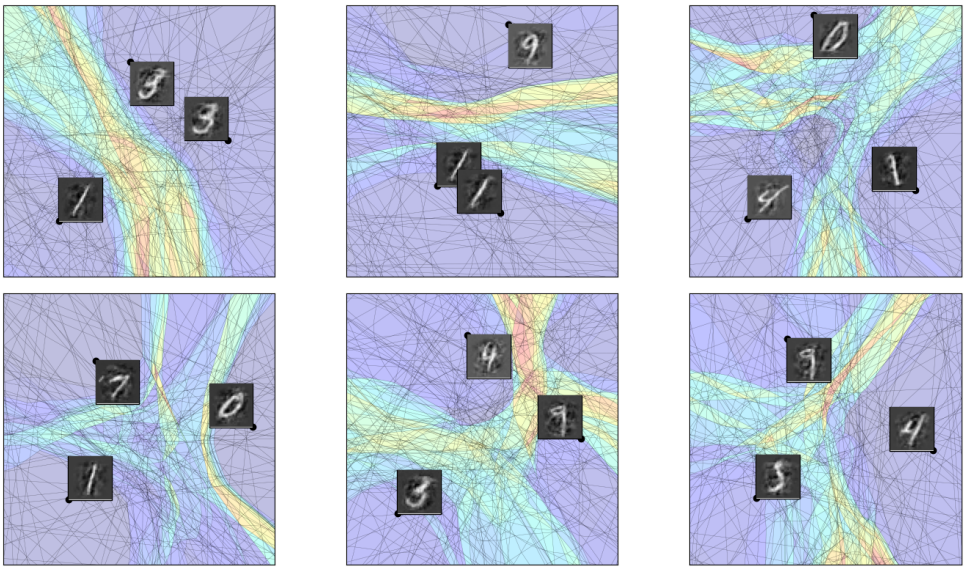


Figure 11: Here we corroborate the right plot of Figure 1 by replicating it on other samples in the input space.

E Proofs of Main Results

Proof of Theorem 2.

1. In the instance of the cross-entropy loss function,

$$\ell_p := \ell\left(f\left(\mathbf{x}_p\right), y_p\right) = -\log\left(\frac{\exp\left(\left[f\left(\mathbf{x}_p\right)\right]_{y_p}\right)}{\sum_{c=1}^{C}\exp\left(\left[f\left(\mathbf{x}_p\right)\right]_c\right)}\right).$$

Under the assumptions, the output of the neural network at $\mathbf{x}_p$ is $A_{\omega_{\mathbf{x}_p}}\mathbf{x}_p$. The cross entropy loss of the deep network on $\mathcal{D}$ is

$$\mathcal{L}_{\text{CE}} = \frac{1}{m}\sum_{p=1}^{m}\ell_p$$

where

$$\begin{aligned}\ell_p &= -\log\left(\frac{\exp\left(\left[A_{\omega_{\mathbf{x}_p}}\mathbf{x}_p\right]_{y_p,\cdot}\right)}{\sum_{c=1}^{C}\exp\left(\left[A_{\omega_{\mathbf{x}_p}}\mathbf{x}_p\right]_{c,\cdot}\right)}\right)\\ &= -\left\langle\left[A_{\omega_{\mathbf{x}_p}}\right]_{y_p,\cdot},\mathbf{x}_p\right\rangle + \log\left(\sum_{c=1}^{C}\exp\left(\left\langle\left[A_{\omega_{\mathbf{x}_p}}\right]_{c,\cdot},\mathbf{x}_p\right\rangle\right)\right),\end{aligned}$$

which is convex on a convex set. Thus we can consider the sufficient Karush-Kuhn-Tucker conditions with Lagrange multiplier,

$$\mathcal{L} = \mathcal{L}_{\text{CE}} + \lambda\left(\sum_{c=1}^{C}\left\|\left[A_{\omega_{\mathbf{x}_p}}\right]_{c,\cdot}\right\|_2^2 - \alpha\right).$$

In particular, the Karush-Kuhn-Tucker conditions have the form

$$\begin{aligned}\frac{\partial\mathcal{L}}{\partial\left[A_{\omega_{\mathbf{x}_p}}\right]_{c,\cdot}} &= -\mathbf{1}_{\{y_p=c\}}\mathbf{x}_p + \frac{\mathbf{x}_p\exp\left(\left\langle\left[A_{\omega_{\mathbf{x}_p}}\right]_{c,\cdot},\mathbf{x}_p\right\rangle\right)}{\sum_{c'=1}^{C}\exp\left(\left\langle\left[A_{\omega_{\mathbf{x}_p}}\right]_{c',\cdot},\mathbf{x}_p\right\rangle\right)} - 2\lambda\left[A_{\omega_{\mathbf{x}_p}}\right]_{c,\cdot}\\ &= \mathbf{0},\end{aligned} \tag{1}$$

for $c = 1,\ldots,C$ and

$$\frac{\partial\mathcal{L}}{\partial\lambda} = \alpha - \sum_{c=1}^{C}\left\|\left[A_{\omega_{\mathbf{x}_p}}\right]_{c,\cdot}\right\|_2^2 = 0. \tag{2}$$

From (1), we have

$$\begin{aligned}0 &= \sum_{c=1}^{C}\left\langle\left[A_{\omega_{\mathbf{x}_p}}\right]_{c,\cdot},\frac{\partial\mathcal{L}}{\partial\left[A_{\omega_{\mathbf{x}_p}}\right]_{c,\cdot}}\right\rangle\\ &= -\left\langle\left[A_{\omega_{\mathbf{x}_p}}\right]_{y_p,\cdot},\mathbf{x}_p\right\rangle + \sum_{c=1}^{C}\frac{\left\langle\left[A_{\omega_{\mathbf{x}_p}}\right]_{c,\cdot},\mathbf{x}_p\right\rangle\exp\left(\left\langle\left[A_{\omega_{\mathbf{x}_p}}\right]_{c,\cdot},\mathbf{x}_p\right\rangle\right)}{\sum_{c'=1}^{C}\exp\left(\left\langle\left[A_{\omega_{\mathbf{x}_p}}\right]_{c',\cdot},\mathbf{x}_p\right\rangle\right)}\\ &\quad - 2\lambda\sum_{c=1}^{C}\left\|\left[A_{\omega_{\mathbf{x}_p}}\right]_{c,\cdot}\right\|_2^2.\end{aligned}$$

Let $\varrho_c = \frac{\exp\left(\left\langle\left[A_{\omega_{\mathbf{x}_p}}\right]_{c,\cdot},\mathbf{x}_p\right\rangle\right)}{\sum_{c'=1}^{C}\exp\left(\left\langle\left[A_{\omega_{\mathbf{x}_p}}\right]_{c',\cdot},\mathbf{x}_p\right\rangle\right)}$. Then, in conjunction with (2), it follows that

$$\lambda = \frac{1}{2\alpha}\sum_{c=1}^{C}\left\langle\left[A_{\omega_{\mathbf{x}_p}}\right]_{c,\cdot},\mathbf{x}_p\right\rangle\varrho_c - \left\langle\left[A_{\omega_{\mathbf{x}_p}}\right]_{y_p,\cdot},\mathbf{x}_p\right\rangle.$$

Using this back in (1) we obtain

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \left[A_{\omega_{\mathbf{x}_p}}\right]_{c,\cdot}} &= -\mathbf{1}_{\{y_p=c\}}\mathbf{x}_p + \mathbf{x}_p\varrho_c + \frac{1}{\alpha}\left\langle \left[A_{\omega_{\mathbf{x}_p}}\right]_{y_p,\cdot}, \mathbf{x}_p \right\rangle \left[A_{\omega_{\mathbf{x}_p}}\right]_{c,\cdot} \\
&\quad - \frac{1}{\alpha}\sum_{c'=1}^{C} \varrho_{c'} \left\langle \left[A_{\omega_{\mathbf{x}_p}}\right]_{c',\cdot}, \mathbf{x}_p \right\rangle \left[A_{\omega_{\mathbf{x}_p}}\right]_{c,\cdot} \\
&= \left(-\mathbf{1}_{\{y_p=c\}} + \varrho_c\right)\mathbf{x}_p + \frac{\left\langle \left[A_{\omega_{\mathbf{x}_p}}\right]_{y_p,\cdot}, \mathbf{x}_p \right\rangle \left[A_{\omega_{\mathbf{x}_p}}\right]_{c,\cdot}}{\alpha}\left(1-\varrho_{y_p}\right) \\
&\quad - \frac{1}{\alpha}\sum_{c'\neq y_p} \left\langle \left[A_{\omega_{\mathbf{x}_p}}\right]_{c',\cdot}, \mathbf{x}_p \right\rangle \left[A_{\omega_{\mathbf{x}_p}}\right]_{c,\cdot} \varrho_{c'} \\
&= \left(-\mathbf{1}_{\{y_p=c\}} + \varrho_c\right)\mathbf{x}_p + \frac{\left\langle \left[A_{\omega_{\mathbf{x}_p}}\right]_{y_p,\cdot}, \mathbf{x}_p \right\rangle \left[A_{\omega_{\mathbf{x}_p}}\right]_{c,\cdot}}{\alpha}\left(1-\varrho_{y_p}\right) \\
&\quad - \frac{1}{\alpha}\left\langle \left[A_{\omega_{\mathbf{x}_p}}\right]_{i,\cdot}, \mathbf{x}_p \right\rangle \left[A_{\omega_{\mathbf{x}_p}}\right]_{c,\cdot}\left(1-\varrho_{y_p}\right) \\
&= \left(-\mathbf{1}_{\{y_p=c\}} + \varrho_c\right)\mathbf{x}_p \\
&\quad + \frac{\left(1-\varrho_{y_p}\right)\left[A_{\omega_{\mathbf{x}_p}}\right]_{c,\cdot}}{\alpha}\left(\left\langle \left[A_{\omega_{\mathbf{x}_p}}\right]_{y_p,\cdot}, \mathbf{x}_p \right\rangle - \left\langle \left[A_{\omega_{\mathbf{x}_p}}\right]_{i,\cdot}, \mathbf{x}_p \right\rangle\right),
\end{aligned}$$

where i is just some incorrect class for $\mathbf{x}_p$, namely $i \neq y_p$. When $c = y_p$ this reduces to

$$\frac{\partial \mathcal{L}}{\partial \left[A_{\omega_{\mathbf{x}_p}}\right]_{c,\cdot}} = \left(1-\varrho_{y_p}\right)\left(-\mathbf{x}_p + \frac{1}{\alpha}\left\langle \left[A_{\omega_{\mathbf{x}_p}}\right]_{y_p,\cdot} - \left[A_{\omega_{\mathbf{x}_p}}\right]_{i,\cdot}, \mathbf{x}_p \right\rangle \left[A_{\omega_{\mathbf{x}_p}}\right]_{y_p,\cdot}\right),$$

and when $c \neq y_p$ it reduces to

$$\frac{\partial \mathcal{L}}{\partial \left[A_{\omega_{\mathbf{x}_p}}\right]_{c,\cdot}} = \left(1-\varrho_{y_p}\right)\left(\frac{1}{C-1}\mathbf{x}_p + \frac{1}{\alpha}\left\langle \left[A_{\omega_{\mathbf{x}_p}}\right]_{y_p,\cdot} - \left[A_{\omega_{\mathbf{x}_p}}\right]_{i,\cdot}, \mathbf{x}_p \right\rangle \left[A_{\omega_{\mathbf{x}_p}}\right]_{c,\cdot}\right).$$

To obtain the optimal A, it suffices to find $A_{\omega_{\mathbf{x}_p}}$ satisfying these conditions. To do so we consider the ansatz

$$\left[A_{\omega_{\mathbf{x}_p}}\right]_{c,\cdot} = \begin{cases} a\mathbf{x}_p & c = y_p \\ b\mathbf{x}_p & c \neq y_p. \end{cases}$$

Substituting this into our conditions we obtain the equations

$$\begin{cases} -1 + \frac{1}{\alpha}(a-b)a\left\|\mathbf{x}_p\right\|_2^2 = 0 \\ \frac{1}{C-1} + \frac{1}{\alpha}(a-b)b\left\|\mathbf{x}_p\right\|_2^2 = 0 \\ \left(a^2 + (C-1)b^2\right)\left\|\mathbf{x}_p\right\|_2^2 = \alpha. \end{cases}$$

Solving these systems of equations we arrive at

$$\begin{cases} a = \frac{1}{\left\|\mathbf{x}_p\right\|_2}\sqrt{\frac{\alpha(C-1)}{C}} \\ b = -\frac{1}{\left\|\mathbf{x}_p\right\|_2}\sqrt{\frac{\alpha}{C(C-1)}}. \end{cases}$$

2. In the instance of the mean-squared error,

$$\ell_p := \ell\left(f\left(\mathbf{x}_p\right), y_p\right) = \left\|f\left(\mathbf{x}_p\right) - \mathbf{e}_{y_p}\right\|_2^2,$$

where we use $\mathbf{e}_i \in \mathbb{R}^d$ to denote the i^{th} standard basis vector. Under the assumptions, the output of the deep network at $\mathbf{x}_p$ is $A_{\omega_{\mathbf{x}_p}}\mathbf{x}_p$. The mean squared error loss of the deep network on $\mathcal{D}$ is

$$\mathcal{L}_{\text{MSE}} = \frac{1}{m}\sum_{p=1}^{m}\ell_p$$

where

$$\ell_p = \left\langle A_{\omega_{\mathbf{x}_p}}\mathbf{x}_p - \mathbf{e}_{y_p}, A_{\omega_{\mathbf{x}_p}}\mathbf{x}_p - \mathbf{e}_{y_p} \right\rangle = \sum_{c=1}^{C}\left(\left\langle \left[A_{\omega_{\mathbf{x}_p}}\right]_{c,\cdot}, \mathbf{x}_p \right\rangle - \mathbf{1}_{\{y_p=c\}}\right)^2,$$

which is a convex function on a convex set. Thus we can consider the sufficient Karush-Kuhn-Tucker conditions with Langrange multiplier,

$$\mathcal{L} = \mathcal{L}_{\text{MSE}} - \lambda\left(\sum_{c=1}^{C}\left\|\left[A_{\omega_{\mathbf{x}_p}}\right]_{c,\cdot}\right\|_2^2 - \alpha\right).$$

In particular, the Karush-Kuhn-Tucker conditions have the form

$$\frac{\partial\mathcal{L}}{\partial\left[A_{\omega_{\mathbf{x}_p}}\right]_{c,\cdot}} = 2\left(\left\langle \left[A_{\omega_{\mathbf{x}_p}}\right]_{c,\cdot}, \mathbf{x}_p \right\rangle - \mathbf{1}_{\{y_p=c\}}\right)\mathbf{x}_p - 2\lambda\left[A_{\omega_{\mathbf{x}_p}}\right]_{c,\cdot}. \tag{3}$$

for $c = 1, \ldots, C$ and

$$\frac{\partial\mathcal{L}}{\partial\lambda} = \sum_{c=1}^{C}\left\|\left[A_{\omega_{\mathbf{x}_p}}\right]_{c,\cdot}\right\|_2^2 - \alpha = 0. \tag{4}$$

From (3), we have

$$\begin{aligned}
0 &= \sum_{c=1}^{C}\left\langle \left[A_{\omega_{\mathbf{x}_p}}\right]_{c,\cdot}, \frac{\partial\mathcal{L}}{\partial\left[A_{\omega_{\mathbf{x}_p}}\right]_{c,\cdot}} \right\rangle \\
&= -2\left\langle \left[A_{\omega_{\mathbf{x}_p}}\right]_{y_p,\cdot}, \mathbf{x}_p \right\rangle + 2\sum_{c=1}^{C}\left\langle \left[A_{\omega_{\mathbf{x}_p}}\right]_{c,\cdot}, \mathbf{x}_p \right\rangle^2 - 2\lambda\sum_{c=1}^{C}\left\|\left[A_{\omega_{\mathbf{x}_p}}\right]_{c,\cdot}\right\|_2^2.
\end{aligned}$$

Then, in conjunction with (4), it follows that

$$\lambda = -\frac{1}{\alpha}\left\langle \left[A_{\omega_{\mathbf{x}_p}}\right]_{y_p,\cdot}, \mathbf{x}_p \right\rangle + \frac{1}{\alpha}\sum_{c=1}^{C}\left\langle \left[A_{\omega_{\mathbf{x}_p}}\right]_{c,\cdot}, \mathbf{x}_p \right\rangle^2.$$

Using this back in (3) we obtain,

$$\begin{aligned}
\frac{\partial\mathcal{L}}{\partial\left[A_{\omega_{\mathbf{x}_p}}\right]_{c,\cdot}} &= 2\left(\left\langle \left[A_{\omega_{\mathbf{x}_p}}\right]_{c,\cdot}, \mathbf{x}_p \right\rangle - \mathbf{1}_{\{y_p=c\}}\right)\mathbf{x}_p \\
&\quad + \left(\frac{2}{\alpha}\left\langle \left[A_{\omega_{\mathbf{x}_p}}\right]_{y_p,\cdot}, \mathbf{x}_p \right\rangle - \frac{2}{\alpha}\sum_{c=1}^{C}\left\langle \left[A_{\omega_{\mathbf{x}_p}}\right]_{c,\cdot}, \mathbf{x}_p \right\rangle^2\right)\left[A_{\omega_{\mathbf{x}_p}}\right]_{c,\cdot}
\end{aligned}$$

To obtain the optimal A, it suffices to find $A_{\omega_{\mathbf{x}_p}}$ satisfying these conditions. To do so we consider the ansatz

$$\left[A_{\omega_{\mathbf{x}_p}}\right]_{c,\cdot} = \begin{cases} a\mathbf{x}_p & c = y_p \\ b\mathbf{x}_p & c \neq y_p. \end{cases}$$

Substituting this into our conditions it follows that $b = 0$ and $a = \frac{\sqrt{\alpha}}{\|\mathbf{x}_p\|_2}$.

□

Proof of Theorem 3. Without loss of generality, we can assume $A_{\omega_{\mathbf{x}}}$ to be of the form $\mathbf{c}\mathbf{v}^\top$ for some $\mathbf{v} \in \mathbb{R}^d$. In particular, we assume that the vector $\mathbf{v}$ is of the same norm as $\mathbf{x}$. Then, locally in $\omega_{\mathbf{x}}$, we have

$$f_\theta(\mathbf{x} + \boldsymbol{\epsilon}) = \mathbf{c}\mathbf{v}^\top(\mathbf{x} + \boldsymbol{\epsilon}).$$

Therefore, $\mathbf{x}$ will only be misclassified by the neural network when $\mathbf{v}^\top(\mathbf{x} + \boldsymbol{\epsilon}) < 0$. From the Cauchy-Schwartz inequality we have that

$$-\|\mathbf{v}\|_2\|\boldsymbol{\epsilon}\|_2 \leq \mathbf{v}^\top\boldsymbol{\epsilon} < -\mathbf{v}^\top\mathbf{x}.$$

Hence,

$$\|\boldsymbol{\epsilon}\|_2 > \frac{\mathbf{v}^\top\mathbf{x}}{\|\mathbf{v}\|_2},$$

the right-hand side of which is maximized when $\mathbf{v}$ is $\mathbf{x}$. □

Proof of Theorem 4. Using Theorem 12 and Theorem 13, it follows that

$$\begin{aligned}\mu_{\omega_{\mathbf{x}}^{(1\leftarrow l)}}^{(1\leftarrow l)} &= \left(A_{\omega_{\mathbf{x}}^{(l-1)}}^{(l-1)} \cdots A_{\omega_{\mathbf{x}}^{(1)}}^{(1)}\right)^{\top} \mu_{\omega_{\mathbf{x}}^{(l)}}^{(l)} \\ &= \left(A_{\omega_{\mathbf{x}}^{(l-1)}}^{(l-1)} \cdots A_{\omega_{\mathbf{x}}^{(1)}}^{(1)}\right)^{\top} \left(A_{\omega_{\mathbf{x}}^{(l)}}^{(l)}\right)^{\top} \mathbf{1} \\ &= \left(A_{\omega_{\mathbf{x}}^{(l)}}^{(l)} \cdots A_{\omega_{\mathbf{x}}^{(1)}}^{(1)}\right)^{\top} \mathbf{1} \\ &= \left(A_{\omega_{\mathbf{x}}^{(1\leftarrow l)}}^{(1\leftarrow l)}\right)^{\top} \mathbf{1}.\end{aligned}$$

Extending this to the L^{th} yields the desired result. □

Proof of Proposition 6. Using Theorem 4, the centroid of an aligned Jacobian is $\mu_{\mathbf{x}} = \mathbf{x}\mathbf{c}^{\top}\mathbf{1} = c\mathbf{x}$ where $c = \mathbf{c}^{\top}\mathbf{1}$. □

Proof of Lemma 7. This follows immediately from the application of Theorem 4. □

Proof of Lemma 8. Observe that in this setting we have

$$\Theta\left(\mathbf{x}, \mathbf{x}'\right) = \left\langle \nabla_{W^{(2)}} f_{\theta}(\mathbf{x}), \nabla_{W^{(2)}} f_{\theta}\left(\mathbf{x}'\right)\right\rangle + \left\langle \nabla_{W^{(1)}} f_{\theta}(\mathbf{x}), \nabla_{W^{(1)}} f_{\theta}\left(\mathbf{x}'\right)\right\rangle.$$

Therefore, noting that

$$\nabla_{W^{(2)}} f_{\theta}(\mathbf{x}) = \sigma\left(W^{(1)}\mathbf{x}\right)$$

and

$$\nabla_{W^{(1)}} f_{\theta}(\mathbf{x}) = W^{(2)} Q[\mathbf{x}]\mathbf{x}^{\top},$$

the result follows. □

Proof of Theorem 9. In a similar way to Proposition 10, one can show that

$$\begin{aligned}\partial_t \mu_{\mathbf{x}} = \frac{\eta}{m}\sum_{p=1}^{m}\Bigg(&\left(m\left[\mathbf{x}_p\right]^{\top} W^{(2)} Q\left[\mathbf{x}_p\right] Q[\mathbf{x}]\left(W^{(2)}\right)^{\top}\mathbf{1}\right)\mathbf{x}_p \\ &+ \left(W^{(1)}\right)^{\top} Q[\mathbf{x}]\sigma\left(W^{(1)}\mathbf{x}_p\right) m\left[\mathbf{x}_p\right]^{\top}\mathbf{1}\Bigg).\end{aligned}$$

Using Lemma 8, this simplifies to

$$\partial_t\left(\langle \mathbf{x}', \mu_{\boldsymbol{\nu}}\rangle\right) = \frac{\eta}{m}\sum_{p=1}^{m}\Theta\left(\mathbf{x}', \mathbf{x}_p\right) m\left[\mathbf{x}_p\right].$$

□

Proof of Proposition 10. From Lemma 7, observe that

$$\partial_t \mu_{\mathbf{x}} = \left(\partial_t\left(W^{(2)}\right) Q[\mathbf{x}]W^{(1)} + W^{(2)} Q[\mathbf{x}]\partial_t\left(W^{(1)}\right)\right)^{\top}\mathbf{1},$$

where

$$\partial_t\left(W^{(i)}\right) = -\eta\nabla_{W^{(i)}}\mathcal{L}$$

for $i = 1, 2$. One can show that

$$\nabla_{W^{(1)}}\mathcal{L} = -\frac{1}{m}\sum_{p=1}^{m}\left(W^{(2)} Q\left[\mathbf{x}_p\right]\right)^{\top}\mathbf{m}\left[\mathbf{x}_p\right]\mathbf{x}_p^{\top}$$

and

$$\nabla_{W^{(2)}}\mathcal{L} = -\frac{1}{m}\sum_{p=1}^{m}\mathbf{m}\left[\mathbf{x}_p\right]\sigma\left(W^{(1)}\mathbf{x}_p\right)^{\top}.$$

Therefore,

$$\partial_t \mu_{\mathbf{x}} = \frac{\eta}{m} \sum_{p=1}^{m} \left(\left(\mathbf{m}\left[\mathbf{x}_p\right]^\top W^{(2)} Q\left[\mathbf{x}_p\right] Q[\mathbf{x}] \left(W^{(2)}\right)^\top \mathbf{1} \right) \mathbf{x}_p + \left(W^{(1)}\right)^\top Q[\mathbf{x}] \sigma\left(W^{(1)} \mathbf{x}_p\right) \mathbf{m}\left[\mathbf{x}_p\right]^\top \mathbf{1} \right).$$

In particular,

$$\mathbf{m}\left[\mathbf{x}_p\right]^\top \mathbf{1} = 1 - \frac{\sum_{c=1}^{C} \exp\left(\left[f_\theta\left(\mathbf{x}_p\right)\right]_c\right)}{\sum_{c'=1}^{C} \exp\left(\left[f_\theta\left(\mathbf{x}_p\right)\right]_{c'}\right)} = 0,$$

meaning

$$\partial_t \mu_{\mathbf{x}} = \frac{\eta}{m} \sum_{p=1}^{m} \left(\mathbf{m}\left[\mathbf{x}_p\right]^\top W^{(2)} Q\left[\mathbf{x}_p\right] Q[\mathbf{x}] \left(W^{(2)}\right)^\top \mathbf{1} \right) \mathbf{x}_p.$$

Therefore, the result follows since $\partial_t \langle \mathbf{x}, \mu_{\mathbf{x}} \rangle = \langle \mathbf{x}, \partial_t \left(\mu_{\mathbf{x}}\right) \rangle$. □

F Supporting Results

Note that, for a continuous piecewise affine network $f = \left(f^{(L)} \circ \cdots \circ f^{(1)}\right)$, each $f^{(l)}$ and sub-component $f^{(1 \leftarrow l)} = \left(f^{(l)} \circ \cdots \circ f^{(1)}\right)$ are also continuous piecewise affine networks. Let $A^{(l)}_{\omega^{(l)}_{\mathbf{x}}}$, $B^{(l)}_{\omega^{(l)}_{\mathbf{x}}}$, $\omega^{(l)}_{\mathbf{x}}$, $\mu^{(\ell)}_{\omega^{(l)}_{\mathbf{x}}}$ and $A^{(1 \leftarrow l)}_{\omega^{(1 \leftarrow l)}_{\mathbf{x}}}$, $B^{(1 \leftarrow l)}_{\omega^{(1 \leftarrow l)}_{\mathbf{x}}}$, $\omega^{(1 \leftarrow l)}_{\mathbf{x}}$, $\mu^{(1 \leftarrow l)}_{\omega^{(1 \leftarrow \ell)}_{\mathbf{x}}}$ be analogous notation for the layer and sub-component networks to that of the continuous piecewise affine networks we introduced in Section 2.

Theorem 12 (Balestriero et al. 3)**.** *The l^{th} layer of a deep network partitions its input space according to a power diagram with centroids*

$$\mu^{(\ell)}_{\omega^{(l)}_{\mathbf{x}}} = \left(A^{(l)}_{\omega_{\mathbf{x}^{(l)}}}\right)^\top \mathbf{1}.$$

Theorem 13 (Balestriero et al. 3)**.** *The continuous piecewise operation of a deep network from the input to the output of the l^{th} layer partitions its input space according to a power diagram with centroids*

$$\mu^{(1 \leftarrow l)}_{\omega^{(1 \leftarrow \ell)}_{\mathbf{x}}} = \left(A^{(l-1)}_{\omega^{(l-1)}_{\mathbf{x}}} \cdots A^{(1)}_{\omega^{(1)}_{\mathbf{x}}}\right)^\top \mu^{(l)}_{\omega^{(l)}_{\mathbf{x}}} =: \left(A^{(1 \leftarrow l-1)}_{\omega^{(1 \leftarrow l-1)}_{\mathbf{x}}}\right)^\top \mu^{(l)}_{\omega^{(l)}_{\mathbf{x}}}.$$

G Compute Resources

Our experiments were computed on a range of NVIDIA GPUs, including GTX TITAN Xs, RTX 2080Tis, and RTX 8000s. In Table 3 we indicate roughly how long each of our main experiments took to run.

Table 3: The computational resources utilised to perform the experiments of this work.

Experiment	Hardware	Time
Figure 2	GTX TITAN X	4 hours
Figure 3	GTX TITAN X	Less than 1 hour
Figure 4	GTX TITAN X	Less than 1 hour
Figures 5 and 6	GTX 1080 Ti	6 hours
Figure 7	GTX TITAN X	4 hours
Figures 8 and 9	GTX 1080 Ti	5 hours
Figure 10	GTX TITAN X	1 day
Tables 1 and 2	GTX 1080 Ti, RTX 8000	5 days