

# Explaining grokking through circuit efficiency

Vikrant Varma<sup>\*, 1</sup>, Rohin Shah<sup>\*, 1</sup>, Zachary Kenton<sup>1</sup>, János Kramár<sup>1</sup> and Ramana Kumar<sup>1</sup>

<sup>\*</sup>Equal contributions, <sup>1</sup>Google DeepMind

One of the most surprising puzzles in neural network generalisation is *grokking*: a network with perfect training accuracy but poor generalisation will, upon further training, transition to perfect generalisation. We propose that grokking occurs when the task admits a generalising solution and a memorising solution, where the generalising solution is slower to learn but more *efficient*, producing larger logits with the same parameter norm. We hypothesise that memorising circuits become more inefficient with larger training datasets while generalising circuits do not, suggesting there is a critical dataset size at which memorisation and generalisation are equally efficient. We make and confirm four novel predictions about grokking, providing significant evidence in favour of our explanation. Most strikingly, we demonstrate two novel and surprising behaviours: *ungrokking*, in which a network regresses from perfect to low test accuracy, and *semi-grokking*, in which a network shows delayed generalisation to partial rather than perfect test accuracy.

## 1. Introduction

When training a neural network, we expect that once training loss converges to a low value, the network will no longer change much. Power et al. (2021) discovered a phenomenon dubbed *grokking* that drastically violates this expectation. The network first “memorises” the data, achieving low and stable training loss with poor generalisation, but with further training transitions to perfect generalisation. We are left with the question: *why does the network’s test performance improve dramatically upon continued training, having already achieved nearly perfect training performance?*

Recent answers to this question vary widely, including the difficulty of representation learning (Liu et al., 2022), the scale of parameters at initialisation (Liu et al., 2023), spikes in loss (“slingshots”) (Thilak et al., 2022), random walks among optimal solutions (Millidge, 2022), and the simplicity of the generalising solution (Nanda et al., 2023, Appendix E). In this paper, we argue that the last explanation is correct, by stating a specific theory in this genre, deriving novel predictions from the theory, and confirming the predictions empirically.

We analyse the interplay between the internal mechanisms that the neural network uses to calculate the outputs, which we loosely call “circuits” (Olah et al., 2020). We hypothesise that there are two families of circuits that both achieve good training performance: one which generalises well ( $C_{\text{gen}}$ ) and one which memorises the training dataset ( $C_{\text{mem}}$ ). The key insight is that *when there are multiple circuits that achieve strong training performance, weight decay prefers circuits with high “efficiency”*, that is, circuits that require less parameter norm to produce a given logit value.

Efficiency answers our question above: if  $C_{\text{gen}}$  is more efficient than  $C_{\text{mem}}$ , gradient descent can reduce nearly perfect training loss even further by strengthening  $C_{\text{gen}}$  while weakening  $C_{\text{mem}}$ , which then leads to a transition in test performance. With this understanding, we demonstrate in Section 3 that three key properties are sufficient for grokking: (1)  $C_{\text{gen}}$  generalises well while  $C_{\text{mem}}$  does not, (2)  $C_{\text{gen}}$  is more efficient than  $C_{\text{mem}}$ , and (3)  $C_{\text{gen}}$  is learned more slowly than  $C_{\text{mem}}$ .

Since  $C_{\text{gen}}$  generalises well, it automatically works for any new data points that are added to the training dataset, and so its efficiency should be independent of the size of the training dataset. In contrast,  $C_{\text{mem}}$  must memorise any additional data points added to the training dataset, and so

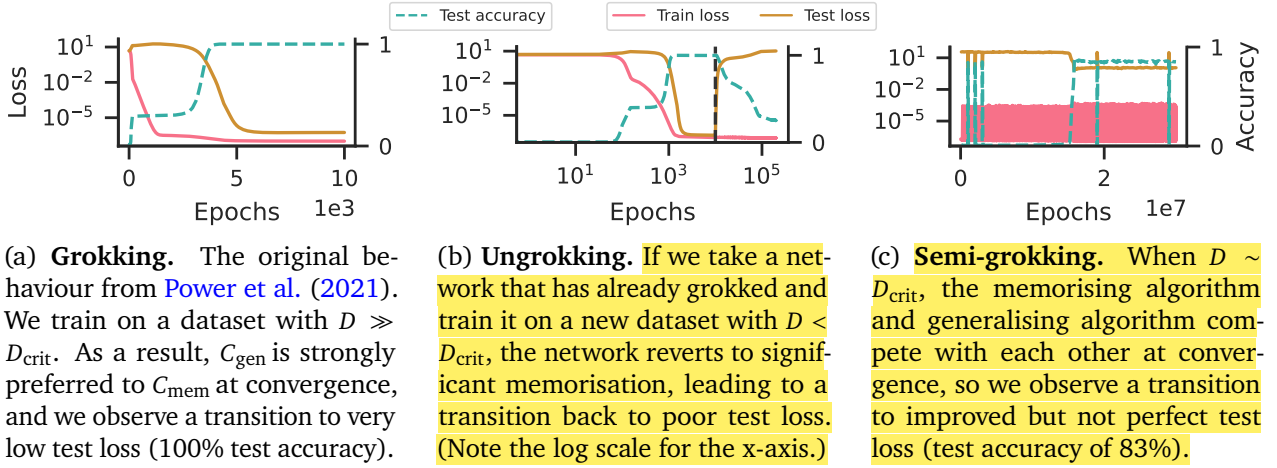


Figure 1 | **Novel grokking phenomena.** When grokking occurs, we expect there are two algorithms that perform well at training: “memorisation” ( $C_{\text{mem}}$ , with poor test performance) and “generalisation” ( $C_{\text{gen}}$ , with strong test performance). Weight decay strengthens  $C_{\text{gen}}$  over  $C_{\text{mem}}$  as the training dataset size increases. By analysing the point at which  $C_{\text{gen}}$  and  $C_{\text{mem}}$  are equally strong (the critical dataset size  $D_{\text{crit}}$ ), we predict and confirm two novel behaviours: *ungrokking* and *semi-grokking*.

its efficiency should decrease as training dataset size increases. We validate these predictions by quantifying efficiencies for various dataset sizes for both  $C_{\text{mem}}$  and  $C_{\text{gen}}$ .

This suggests that there exists a crossover point at which  $C_{\text{gen}}$  becomes more efficient than  $C_{\text{mem}}$ , which we call the critical dataset size  $D_{\text{crit}}$ . By analysing dynamics at  $D_{\text{crit}}$ , we predict and demonstrate two new behaviours (Figure 1). In *ungrokking*, a model that has successfully grokked returns to poor test accuracy when further trained on a dataset much smaller than  $D_{\text{crit}}$ . In *semi-grokking*, we choose a dataset size where  $C_{\text{gen}}$  and  $C_{\text{mem}}$  are similarly efficient, leading to a phase transition but only to middling test accuracy.

We make the following contributions:

1. We demonstrate the sufficiency of three ingredients for grokking through a constructed simulation (Section 3).
2. By analysing dynamics at the “critical dataset size” implied by our theory, we *predict* two novel behaviours: *semi-grokking* and *ungrokking* (Section 4).
3. We confirm our predictions through careful experiments, including demonstrating semi-grokking and ungrokking in practice (Section 5).

## 2. Notation

We consider classification using deep neural networks under the cross-entropy loss. In particular, we are given a set of inputs  $X$ , a set of labels  $Y$ , and a training dataset,  $\mathcal{D} = \{(x_1, y_1^*), \dots, (x_D, y_D^*)\}$ .

For an arbitrary classifier  $h : X \times Y \rightarrow \mathbb{R}$ , the softmax cross entropy loss is given by:

$$\mathcal{L}_{\text{x-ent}}(h) = -\frac{1}{D} \sum_{(x, y^*) \in \mathcal{D}} \log \frac{\exp(h(x, y^*))}{\sum_{y' \in Y} \exp(h(x, y'))}. \quad (1)$$

The output of a classifier for a specific class is the class *logit*, denoted by  $o_h^y(x) := h(x, y)$ . When the

input  $x$  is clear from context, we will denote the logit as  $o_h^y$ . We denote the vector of the logits for all classes for a given input as  $\vec{o}_h(x)$  or  $\vec{o}_h$  when  $x$  is clear from context.

Parametric classifiers (such as neural networks) are parameterised with a vector  $\theta$  that induces a classifier  $h_\theta$ . The parameter norm of the classifier is  $P_{h_\theta} := \|\theta\|$ . It is common to add *weight decay* regularisation, which is an additional loss term  $\mathcal{L}_{\text{wd}}(h_\theta) = \frac{1}{2}(P_{h_\theta})^2$ . The overall loss is given by

$$\mathcal{L}(h_\theta) = \mathcal{L}_{\text{x-ent}}(h) + \alpha \mathcal{L}_{\text{wd}}(h_\theta), \quad (2)$$

where  $\alpha$  is a constant that trades off between softmax cross entropy and weight decay.

**Circuits.** Inspired by [Olah et al. \(2020\)](#), we use the term *circuit* to refer to an internal mechanism by which a neural network works. We only consider circuits that map inputs to logits, so that a circuit  $C$  induces a classifier  $h_C$  for the overall task. We elide this distinction and simply write  $C$  to refer to  $h_C$ , so that the logits are  $o_C^y$ , the loss is  $\mathcal{L}(C)$ , and the parameter norm is  $P_C$ .

For any given algorithm, there exist multiple circuits that implement that algorithm. Abusing notation, we use  $C_{\text{gen}} (C_{\text{mem}})$  to refer either to the *family* of circuits that implements the generalising (memorising) algorithm, or a single circuit from the appropriate family.

### 3. Three ingredients for grokking

Given a circuit with perfect training accuracy (as with a pure memorisation approach like  $C_{\text{mem}}$  or a perfectly generalising solution like  $C_{\text{gen}}$ ), the cross entropy loss  $\mathcal{L}_{\text{x-ent}}$  incentivises gradient descent to scale up the classifier’s logits, as that makes its answers more confident, leading to lower loss (see Theorem D.1). For typical neural networks, this would be achieved by making the parameters larger. Meanwhile, weight decay  $\mathcal{L}_{\text{wd}}$  pushes in the opposite direction, directly decreasing the parameters. These two forces must be balanced at any local minimum of the overall loss.

When we have *multiple* circuits that achieve strong training accuracy, this constraint applies to each individually. But how will they relate to each other? Intuitively, the answer depends on the *efficiency* of each circuit, that is, the extent to which the circuit can convert relatively small parameters into relatively large logits. For more efficient circuits, the  $\mathcal{L}_{\text{x-ent}}$  force towards larger parameters is stronger, and the  $\mathcal{L}_{\text{wd}}$  force towards smaller parameters is weaker. So, we expect that more efficient circuits will be stronger at any local minimum.

Given this notion of efficiency, we can explain grokking as follows. In the first phase,  $C_{\text{mem}}$  is learned quickly, leading to strong train performance and poor test performance. In the second phase,  $C_{\text{gen}}$  is now learned, and parameter norm is “reallocated” from  $C_{\text{mem}}$  to  $C_{\text{gen}}$ , eventually leading to a mixture of strong  $C_{\text{gen}}$  and weak  $C_{\text{mem}}$ , causing an increase in test performance.

This overall explanation relies on the presence of three ingredients:

1. **Generalising circuit:** There are two families of circuits that achieve good training performance: a memorising family  $C_{\text{mem}}$  with poor test performance, and a generalising family  $C_{\text{gen}}$  with good test performance.
2. **Efficiency:**  $C_{\text{gen}}$  is more “efficient” than  $C_{\text{mem}}$ , that is, it can produce equivalent cross-entropy loss on the training set with a lower parameter norm.
3. **Slow vs fast learning:**  $C_{\text{gen}}$  is learned more slowly than  $C_{\text{mem}}$ , such that during early phases of training  $C_{\text{mem}}$  is stronger than  $C_{\text{gen}}$ .

To illustrate the sufficiency of these ingredients, we construct a minimal example containing all three ingredients, and demonstrate that it leads to grokking. We emphasise that this example is to be

treated as a validation of the three ingredients, rather than as a quantitative prediction of the dynamics of existing examples of grokking. Many of the assumptions and design choices were made on the basis of simplicity and analytical tractability, rather than a desire to reflect examples of grokking in practice. The clearest difference is that  $C_{\text{gen}}$  and  $C_{\text{mem}}$  are modelled as hardcoded input-output lookup tables whose outputs can be strengthened through learned scalar weights, whereas in existing examples of grokking  $C_{\text{gen}}$  and  $C_{\text{mem}}$  are learned internal mechanisms in a neural network that can be strengthened by scaling up the parameters implementing those mechanisms.

**Generalisation.** To model generalisation, we introduce a training dataset  $\mathcal{D}$  and a test dataset  $\mathcal{D}_{\text{test}}$ .  $C_{\text{gen}}$  is a lookup table that produces logits that achieve perfect train and test accuracy.  $C_{\text{mem}}$  is a lookup table that achieves perfect train accuracy, but makes confident incorrect predictions on the test dataset. We denote by  $\mathcal{D}_{\text{mem}}$  the predictions made by  $C_{\text{mem}}$  on the test inputs, with the property that there is no overlap between  $\mathcal{D}_{\text{test}}$  and  $\mathcal{D}_{\text{mem}}$ . Then we have:

$$\begin{aligned} o_G^y(x) &= \mathbb{1} [(x, y) \in \mathcal{D} \text{ or } (x, y) \in \mathcal{D}_{\text{test}}] \\ o_M^y(x) &= \mathbb{1} [(x, y) \in \mathcal{D} \text{ or } (x, y) \in \mathcal{D}_{\text{mem}}] \end{aligned}$$

**Slow vs fast learning.** To model learning, we introduce *weights* for each of the circuits, and use gradient descent to update the weights. Thus, the overall logits are given by:

$$o^y(x) = w_G o_G^y(x) + w_M o_M^y(x)$$

Unfortunately, if we learn  $w_G$  and  $w_M$  directly with gradient descent, we have no control over the *speed* at which the weights are learned. Inspired by [Jermyn and Shlegeris \(2022\)](#), we instead compute weights as multiples of two “subweights”, and then learn the subweights with gradient descent. More precisely, we let  $w_G = w_{G_1} w_{G_2}$  and  $w_M = w_{M_1} w_{M_2}$ , and update each subweight according to  $w_i \leftarrow w_i - \lambda \cdot \partial \mathcal{L} / \partial w_i$ . The speed at which the weights are strengthened by gradient descent can then be controlled by the initial values of the weights. Intuitively, the gradient towards the first subweight  $\partial \mathcal{L} / \partial w_1$  depends on the strength of the second subweight  $w_2$  and vice-versa, and so low initial values lead to slow learning. At initialisation, we set  $w_{G_1} = w_{M_1} = 0$  to ensure the logits are initially zero, and then set  $w_{G_2} \ll w_{M_2}$  to ensure  $C_{\text{gen}}$  is learned more slowly than  $C_{\text{mem}}$ .

**Efficiency.** Above, we operationalised circuit efficiency as the extent to which the circuit can convert relatively small parameters into relatively large logits. When the weights are all one, each circuit produces a one-hot vector as its logits, so their logit scales are the same, and efficiency is determined solely by parameter norm. We define  $P_G$  and  $P_M$  to be the parameter norms when weights are all one. Since we want  $C_{\text{gen}}$  to be more efficient than  $C_{\text{mem}}$ , we set  $P_G < P_M$ .

This still leaves the question of how to model parameter norm when the weights are not all one. Intuitively, increasing the weights corresponds to increasing the parameters in a neural network to scale up the resulting outputs. In a  $\kappa$ -layer MLP with Relu activations and without biases, scaling all parameters by a constant  $c$  scales the outputs by  $c^\kappa$ . Inspired by this observation, we model the parameter norm of  $w_G C_G$  as  $w_G^{1/\kappa} P_i$  for some  $\kappa > 0$ , and similarly for  $w_M C_M$ .

**Theoretical analysis.** We first analyse the optimal solutions to the setup above. We can ignore the subweights, as they only affect the speed of learning:  $\mathcal{L}_{\text{x-ent}}$  and  $\mathcal{L}_{\text{wd}}$  depend only on the weights, not subweights. Intuitively, to get minimal loss, we must assign higher weights to more efficient circuits – but it is unclear whether we should assign *no* weight to less efficient circuits, or merely smaller but still non-zero weights. Theorem [D.4](#) shows that in our example, both of these cases can arise: which one we get depends on the value of  $\kappa$ .

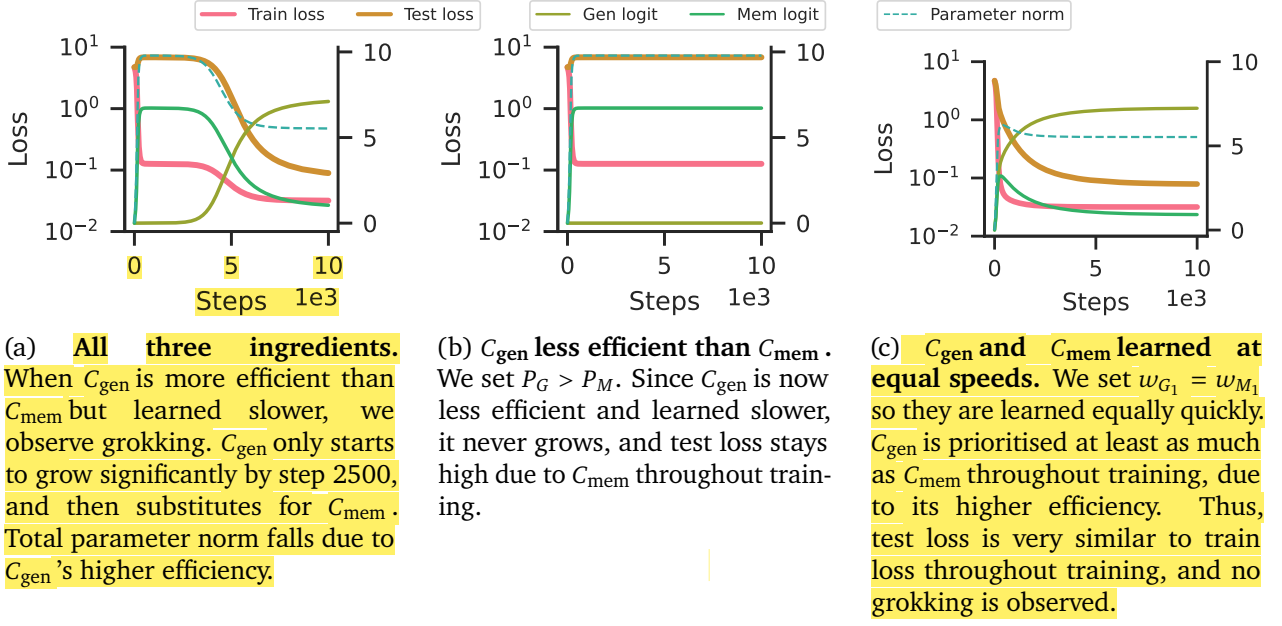


Figure 2 |  $C_{\text{gen}}$  must be learned slowly for grokking to arise. We learn weights  $w_M$  and  $w_G$  through gradient descent on the loss in Equation D.1. To model the fact that  $C_{\text{gen}}$  is more efficient than  $C_{\text{mem}}$ , we set  $P_M > P_G$ . We see that we only get grokking when  $C_{\text{gen}}$  is learned more slowly than  $C_{\text{mem}}$ .

**Experimental analysis.** We run our example for various hyperparameters, and plot training and test loss in Figure 2. We see that when all three ingredients are present (Figure 2a), we observe the standard grokking curves, with a delayed decrease in test loss. By contrast, when we make the generalising circuit less efficient (Figure 2b), the test loss never falls, and when we remove the slow vs fast learning ingredient (Figure 2c), we see that test loss decreases immediately. See Appendix C for details.

## 4. Why generalising circuits are more efficient

Section 3 demonstrated that grokking can arise when  $C_{\text{gen}}$  is more efficient than  $C_{\text{mem}}$ , but left open the question of *why*  $C_{\text{gen}}$  is more efficient. In this section, we develop a theory based on training dataset size  $D$ , and use it to predict two new behaviours: *ungrokking* and *semi-grokking*.

### 4.1. Relationship of efficiency with dataset size

Consider a classifier  $h_{\mathcal{D}}$  obtained by training on a dataset  $\mathcal{D}$  of size  $D$  with weight decay, and a classifier  $h_{\mathcal{D}'}$  obtained by training on the same dataset with one additional point:  $\mathcal{D}' = \mathcal{D} \cup \{(x, y^*)\}$ . Intuitively,  $h_{\mathcal{D}'}$  cannot be *more* efficient than  $h_{\mathcal{D}}$ : if it was, then  $h_{\mathcal{D}'}$  would outperform  $h_{\mathcal{D}}$  even on just  $\mathcal{D}$ , since it would get similar  $\mathcal{L}_{\text{x-ent}}$  while doing better by weight decay. So we should expect that, on average, classifier efficiency is monotonically non-increasing in dataset size.

How does generalisation affect this picture? Let us suppose that  $h_{\mathcal{D}}$  successfully generalises to predict  $y^*$  for the new input  $x$ . Then, as we move from  $\mathcal{D}$  to  $\mathcal{D}'$ ,  $\mathcal{L}_{\text{x-ent}}(h_{\mathcal{D}})$  likely does not worsen with this new data point. Thus, we could expect to see the same classifier arise, with the same average logit value, parameter norm, and efficiency.

Now suppose  $h_{\mathcal{D}}$  instead *fails* to predict the new data point  $(x, y^*)$ . Then the classifier learned for



$D'$  will likely be *less* efficient:  $\mathcal{L}_{\text{x-ent}}(h_{\mathcal{D}})$  would be much higher due to this new data point, and so the new classifier must incur some additional regularisation loss to reduce  $\mathcal{L}_{\text{x-ent}}$  on the new point.

Applying this analysis to our circuits, we should expect  $C_{\text{gen}}$ 's efficiency to remain unchanged as  $D$  increases arbitrarily high, since  $C_{\text{gen}}$  does need not to change to accommodate new training examples. In contrast,  $C_{\text{mem}}$  must change with nearly every new data point, and so we should expect its efficiency to decrease as  $D$  increases. Thus, when  $D$  is sufficiently large, we expect  $C_{\text{gen}}$  to be more efficient than  $C_{\text{mem}}$ . (Note however that when the set of possible inputs is small, even the maximal  $D$  may not be “sufficiently large”.)

**Critical threshold for dataset size.** Intuitively, we expect that for extremely small datasets (say,  $D < 5$ ), it is extremely easy to memorise the training dataset. So, we hypothesise that for these very small datasets,  $C_{\text{mem}}$  is more efficient than  $C_{\text{gen}}$ . However, as argued above,  $C_{\text{mem}}$  will get less efficient as  $D$  increases, and so there will be a critical dataset size  $D_{\text{crit}}$  at which  $C_{\text{mem}}$  and  $C_{\text{gen}}$  are approximately equally efficient. When  $D \gg D_{\text{crit}}$ ,  $C_{\text{gen}}$  is more efficient and we expect grokking, and when  $D \ll D_{\text{crit}}$ ,  $C_{\text{mem}}$  is more efficient and so grokking should not happen.

**Effect of weight decay on  $D_{\text{crit}}$ .** Since  $D_{\text{crit}}$  is determined only by the relative efficiencies of  $C_{\text{gen}}$  and  $C_{\text{mem}}$ , and none of these depends on the exact value of weight decay (just on weight decay being present at all), our theory predicts that  $D_{\text{crit}}$  should *not* change as a function of weight decay. Of course, the strength of weight decay may still affect other properties such as the number of epochs till grokking.

#### 4.2. Implications of crossover: ungrokking and semi-grokking.

By thinking through the behaviour around the critical threshold for dataset size, we predict the existence of two phenomena that, to the best of our knowledge, have not previously been reported.

**Ungrokking.** Suppose we take a network that has been trained on a dataset with  $D > D_{\text{crit}}$  and has already exhibited grokking, and continue to train it on a smaller dataset with size  $D' < D_{\text{crit}}$ . In this new training setting,  $C_{\text{mem}}$  is now more efficient than  $C_{\text{gen}}$ , and so we predict that with enough further training gradient descent will reallocate weight from  $C_{\text{gen}}$  to  $C_{\text{mem}}$ , leading to a transition from high test performance to low test performance. Since this is exactly the opposite observation as in regular grokking, we term this behaviour “ungrokking”.

Ungrokking can be seen as a special case of catastrophic forgetting (McCloskey and Cohen, 1989; Ratcliff, 1990), where we can make much more precise predictions. First, since ungrokking should only be expected once  $D' < D_{\text{crit}}$ , if we vary  $D'$  we predict that there will be a sharp transition from very strong to near-random test accuracy (around  $D_{\text{crit}}$ ). Second, we predict that ungrokking would arise even if we only remove examples from the training dataset, whereas catastrophic forgetting typically involves training on new examples as well. Third, since  $D_{\text{crit}}$  does not depend on weight decay, we predict the amount of “forgetting” (i.e. the test accuracy at convergence) also does not depend on weight decay.

**Semi-grokking.** Suppose we train a network on a dataset with  $D \approx D_{\text{crit}}$ .  $C_{\text{gen}}$  and  $C_{\text{mem}}$  would be similarly efficient, and there are two possible cases for what we expect to observe (illustrated in Theorem D.4).

In the first case, gradient descent would select either  $C_{\text{mem}}$  or  $C_{\text{gen}}$ , and then make it the maximal circuit. This could happen in a consistent manner (for example, perhaps since  $C_{\text{mem}}$  is learned faster it always becomes the maximal circuit), or in a manner dependent on the random initialisation. In either case we would simply observe the presence or absence of grokking.

In the second case, gradient descent would produce a mixture of both  $C_{\text{mem}}$  and  $C_{\text{gen}}$ . Since neither  $C_{\text{mem}}$  nor  $C_{\text{gen}}$  would dominate the prediction on the test set, we would expect middling test performance.

$C_{\text{mem}}$  would still be learned faster, and so this would look similar to grokking: an initial phase with good train performance and bad test performance, followed by a transition to significantly improved test performance. Since we only get to middling generalisation unlike in typical grokking, we call this behaviour *semi-grokking*.

Our theory does not say which of the two cases will arise in practice, but in Section 5.3 we find that semi-grokking does happen in our setting.

## 5. Experimental evidence

Our explanation of grokking has some support from prior work:

1. **Generalising circuit:** Nanda et al. (2023, Figure 1) identify and characterise the generalising circuit learned at the end of grokking in the case of modular addition.
2. **Slow vs fast learning:** Nanda et al. (2023, Figure 7) demonstrate “progress measures” showing that the generalising circuit develops and strengthens long after the network achieves perfect training accuracy in modular addition.

To further validate our explanation, we empirically test our predictions from Section 4:

- (P1) **Efficiency:** We confirm our prediction that  $C_{\text{gen}}$  efficiency is independent of dataset size, while  $C_{\text{mem}}$  efficiency decreases as training dataset size increases.
- (P2) **Ungrokking (phase transition):** We confirm our prediction that ungrokking shows a phase transition around  $D_{\text{crit}}$ .
- (P3) **Ungrokking (weight decay):** We confirm our prediction that the final test accuracy after ungrokking is independent of the strength of weight decay.
- (P4) **Semi-grokking:** We demonstrate that semi-grokking occurs in practice.

**Training details.** We train 1-layer Transformer models with the AdamW optimiser (Loshchilov and Hutter, 2019) on cross-entropy loss (see Appendix A for more details). All results in this section are on the modular addition task ( $a + b \bmod P$  for  $a, b \in (0, \dots, P - 1)$  and  $P = 113$ ) unless otherwise stated; results on 9 additional tasks can be found in Appendix A.

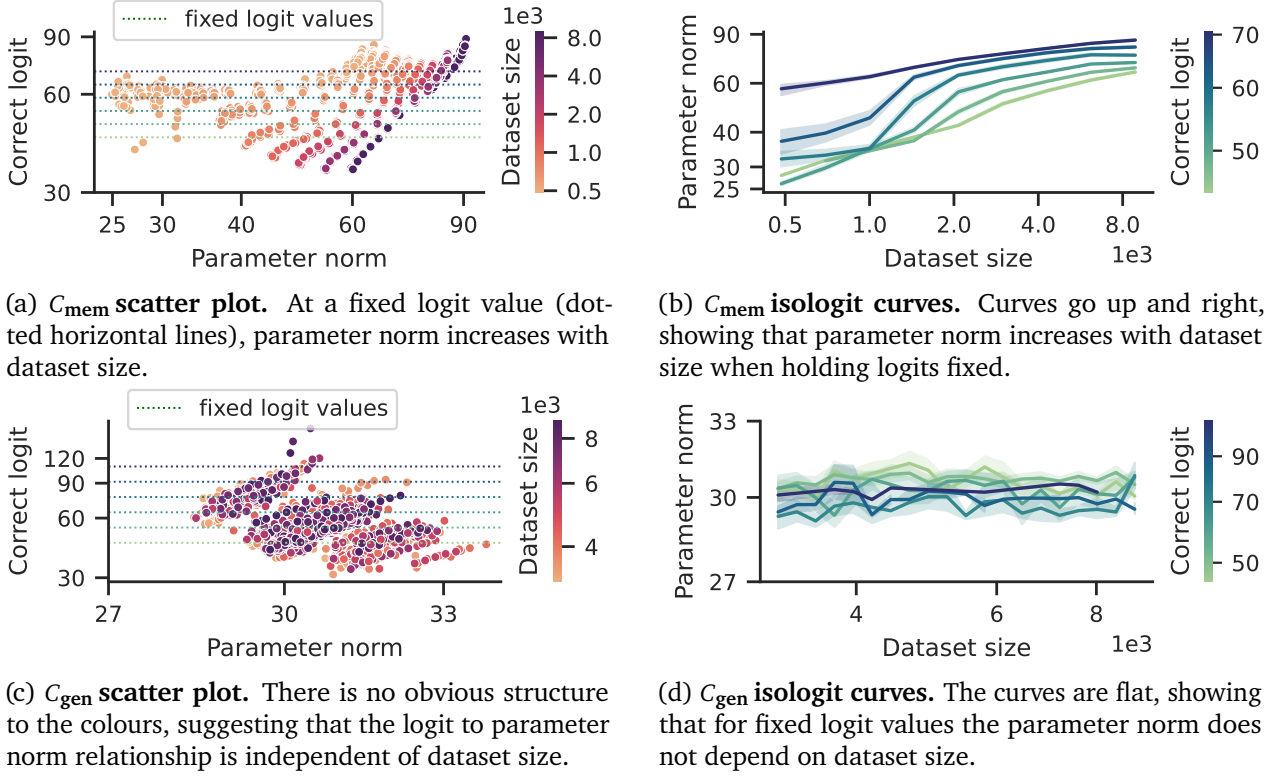
### 5.1. Relationship of efficiency with dataset size

We first test our prediction about memorisation and generalisation efficiency:

(P1) **Efficiency.** We predict (Section 4.1) that memorisation efficiency decreases with increasing train dataset size, while generalisation efficiency stays constant.

To test (P1), we look at training runs where only one circuit is present, and see how the logits  $\sigma_i^y$  vary with the parameter norm  $P_i$  (by varying the weight decay) and the dataset size  $D$ .

**Experiment setup.** We produce  $C_{\text{mem}}$ -only networks by using completely random labels for the training data (Zhang et al., 2021), and assume that the entire parameter norm at convergence is allocated to memorisation. We produce  $C_{\text{gen}}$ -only networks by training on large dataset sizes and checking that  $> 95\%$  of the logit norm comes from just the trigonometric subspace (see Appendix B for details).



**Figure 3 | Efficiency of the  $C_{\text{mem}}$  and  $C_{\text{gen}}$  algorithms.** We collect and visualise a dataset of triples  $(o^y, P_m, D)$  (correct logit, parameter norm, and dataset size), each corresponding to a training run with varying random seed, weight decay, and dataset size, for both  $C_{\text{mem}}$  and  $C_{\text{gen}}$ . Besides a standard scatter plot, we geometrically bucket logit values into six buckets, and plot “isologit curves” showing the dependence of parameter norm on dataset size for each bucket. The results validate our theory that (1)  $C_{\text{mem}}$  requires larger parameter norm to produce the same logits as dataset size increases, and (2)  $C_{\text{gen}}$  uses the same parameter norm to produce fixed logits, irrespective of dataset size. In addition,  $C_{\text{mem}}$  has a much wider range of parameter norms than  $C_{\text{gen}}$ , and at the extremes can be more efficient than  $C_{\text{gen}}$ .

**Results.** Figures 3a and 3b confirm our theoretical prediction for memorisation efficiency. Specifically, to produce a fixed logit value, a higher parameter norm is required when dataset size is increased, implying decreased efficiency. In addition, for a fixed dataset size, scaling up logits requires scaling up parameter norm, as expected. Figures 3c and 3d confirm our theoretical prediction for generalisation efficiency. To produce a fixed logit value, the same parameter norm is required irrespective of the dataset size.

Note that the figures show significant variance across random seeds. We speculate that there are many different circuits implementing the same overall algorithm, but they have different efficiencies, and the random initialisation determines which one gradient descent finds. For example, in the case of modular addition, the generalising algorithm depends on a set of “key frequencies” (Nanda et al., 2023); different choices of key frequencies could lead to different efficiencies.

It may appear from Figure 3c that increasing parameter norm does not increase logit value, contradicting our theory. However, this is a statistical artefact caused by the variance from the random seed. We *do* see “stripes” of particular colours going up and right: these correspond to runs with the same seed and dataset size, but different weight decay, and they show that when the noise from the random seed is removed, increased parameter norm clearly leads to increased logits.



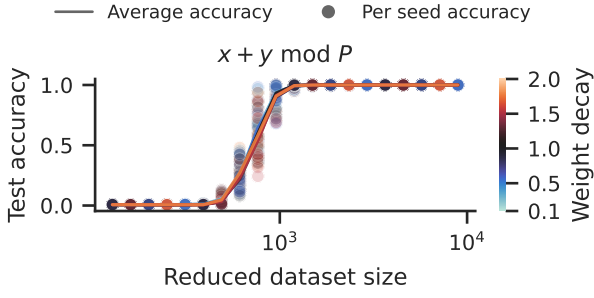


Figure 4 | **Ungrokking.** We train on the full dataset (achieving 100% test accuracy), and then continue training on a smaller subset of the full dataset. We plot test accuracy against reduced dataset size for a range of weight decays. We see a sharp transition from strong test accuracy to near-zero test accuracy, that is independent of weight decay (different coloured lines almost perfectly overlap). See Figure 8 for more tasks.

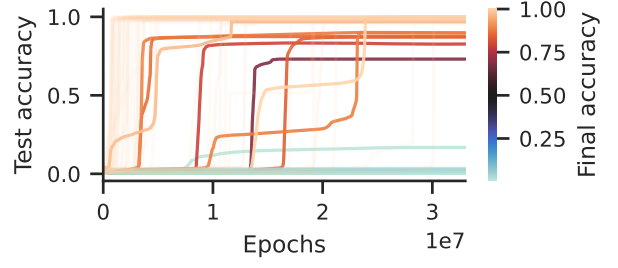


Figure 5 | **Semi-grokking.** We plot test accuracy against training epochs for a large sweep of training runs with varying dataset sizes. Lines are coloured by the final test accuracy at the end of training. Out of 200 runs, at least 6 show clear semi-grokking at the end of training. Many other runs show transient semi-grokking, hovering around middling test accuracy for millions of epochs, or having multiple plateaus, before fully generalising.

## 5.2. Ungrokking: overfitting after generalisation

We now turn to testing our predictions about ungrokking. Figure 1b demonstrates that ungrokking happens in practice. In this section we focus on testing that it has the properties we expect.

**(P2) Phase transition.** We predict (Section 4.2) that if we plot test accuracy at convergence against the size of the reduced training dataset  $D'$ , there will be a phase transition around  $D_{\text{crit}}$ .

**(P3) Weight decay.** We predict (Section 4.2) that test accuracy at convergence is independent of the strength of weight decay.

**Experiment setup.** We train a network to convergence on the full dataset to enable perfect generalisation, then continue training the model on a small subset of the full dataset, and measure the test accuracy at convergence. We vary both the size of the small subset, as well as the strength of the weight decay.

**Results.** Figure 4 shows the results, and clearly confirms both (P2) and (P3). Appendix A has additional results, and in particular Figure 8 replicates the results for many additional tasks.

## 5.3. Semi-grokking: evenly matched circuits

Unlike the previous predictions, semi-grokking is not strictly implied by our theory. However, as we will see, it turns out that it does occur in practice.

**(P4) Semi-grokking.** When training at around  $D \approx D_{\text{crit}}$ , where  $C_{\text{mem}}$  and  $C_{\text{gen}}$  have roughly equal efficiencies, the final network at convergence should either be entirely composed of the most efficient circuit, or of roughly equal proportions of  $C_{\text{mem}}$  and  $C_{\text{gen}}$ . If the latter, we should observe a transition to middling test accuracy well after near-perfect train accuracy.

There are a number of difficulties in demonstrating an example of semi-grokking in practice. First, the time to grok increases super-exponentially as the dataset size  $D$  decreases (Power et al., 2021, Figure 1), and  $D_{\text{crit}}$  is significantly smaller than the smallest dataset size at which grokking

has been demonstrated. Second, the random seed causes significant variance in the efficiency of  $C_{\text{gen}}$  and  $C_{\text{mem}}$ , which in turn affects  $D_{\text{crit}}$  for that run. Third, accuracy changes sharply with the  $C_{\text{gen}}$  to  $C_{\text{mem}}$  ratio (Appendix A). To observe a transition to middling accuracy, we need to have balanced  $C_{\text{gen}}$  and  $C_{\text{mem}}$  outputs, but this is difficult to arrange due to the variance with random seed. To address these challenges, we run many different training runs, on dataset sizes slightly *above* our best estimate of the typical  $D_{\text{crit}}$ , such that some of the runs will (through random noise) have an unusually inefficient  $C_{\text{gen}}$  or an unusually efficient  $C_{\text{mem}}$ , such that the efficiencies match and there is a chance to semi-grok.

**Experiment setup.** We train 10 seeds for each of 20 dataset sizes evenly spaced in the range [1500, 2050] (somewhat above our estimate of  $D_{\text{crit}}$ ).

**Results.** Figure 1c shows an example of a single run that demonstrates semi-grokking, and Figure 5 shows test accuracies over time for every run. These validate our initial hypothesis that semi-grokking may be possible, but also raise new questions.

In Figure 1c, we see two phenomena peculiar to semi-grokking: (1) test accuracy “spikes” several times throughout training before finally converging, and (2) training loss fluctuates in a set range. We leave investigation of these phenomena to future work.

In Figure 5, we observe that there is often *transient* semi-grokking, where a run hovers around middling test accuracy for millions of epochs, or has multiple plateaus, before generalising perfectly. We speculate that each transition corresponds to gradient descent strengthening a new generalising circuit that is more efficient than any previously strengthened circuit, but took longer to learn. We would guess that if we had trained for longer, many of the semi-grokking runs would exhibit full grokking, and many of the runs that didn’t generalise at all would generalise at least partially to show semi-grokking.

Given the difficulty of demonstrating semi-grokking, we only run this experiment on modular addition. However, our experience with modular addition shows that if we only care about values at convergence, we can find them much faster by ungrokking from a grokked network (instead of semi-grokking from a randomly initialised network). Thus the ungrokking results on other tasks (Figure 8) provide some support that we would see semi-grokking on those tasks as well.

## 6. Related work

**Grokking.** Since Power et al. (2021) discovered grokking, many works have attempted to understand why it happens. Thilak et al. (2022) suggest that “slingshots” could be responsible for grokking, particularly in the absence of weight decay, and Notsawo Jr et al. (2023) discuss a similar phenomenon of “oscillations”. In contrast, our explanation applies even where there are no slingshots or oscillations (as in most of our experiments). Liu et al. (2022) show that for a specific (non-modular) addition task with an inexpressive architecture, perfect generalisation occurs when there is enough data to determine the appropriate structured representation. However, such a theory does not explain semi-grokking. We argue that for more typical tasks on which grokking occurs, the critical factor is instead the relative efficiencies of  $C_{\text{mem}}$  and  $C_{\text{gen}}$ . Liu et al. (2023) show that grokking occurs even in non-algorithmic tasks when parameters are initialised to be very large, because memorisation happens quickly but it takes longer for regularisation to reduce parameter norm to the “Goldilocks zone” where generalisation occurs. This observation is consistent with our theory: in non-algorithmic tasks, we expect there exist efficient, generalising circuits, and the increased parameter norm creates the final ingredient (slow learning), leading to grokking. However, we expect that in algorithmic tasks such as the ones we study, the slow learning is caused by some factor other than large parameter

norm at initialisation.

[Davies et al. \(2023\)](#) is the most related work. The authors identify three ingredients for grokking that mirror ours: a *generalising* circuit that is *slowly learned* but is *favoured by inductive biases*, a perspective also mirrored in [Nanda et al. \(2023, Appendix E\)](#). We operationalise the “inductive bias” ingredient as *efficiency* at producing large logits with small parameter norm, and to provide significant empirical support by predicting and verifying the existence of the critical threshold  $D_{\text{crit}}$  and the novel behaviours of semi-grokking and ungrokking.

[Nanda et al. \(2023\)](#) identify the trigonometric algorithm by which the networks solve modular addition after grokking, and show that it grows smoothly over training, and [Chughtai et al. \(2023\)](#) extend this result to arbitrary group compositions. We use these results to define metrics for the strength of the different circuits (Appendix B) which we used in preliminary investigations and for some results in Appendix A (Figures 9 and 10). [Merrill et al. \(2023\)](#) show similar results on sparse parity: in particular, they show that a sparse subnetwork is responsible for the well-generalising logits, and that it grows as grokking happens.

**Weight decay.** While it is widely known that weight decay can improve generalisation ([Krogh and Hertz, 1991](#)), the mechanisms for this effect are multifaceted and poorly understood ([Zhang et al., 2018](#)). We propose a mechanism that is, to the best of our knowledge, novel: generalising circuits tend to be more efficient than memorising circuits at large dataset sizes, and so weight decay preferentially strengthens the generalising circuits.

**Understanding deep learning through circuit-based analysis.** One goal of *interpretability* is to understand the internal mechanisms by which neural networks exhibit specific behaviours, often through the identification and characterisation of specific parts of the network, especially computational subgraphs (“circuits”) that implement human-interpretable algorithms ([Cammarata et al., 2021](#); [Elhage et al., 2021](#); [Erhan et al., 2009](#); [Geva et al., 2020](#); [Li et al., 2022](#); [Meng et al., 2022](#); [Olah et al., 2020](#); [Wang et al., 2022](#)). Such work can also be used to understand deep learning.

[Olsson et al. \(2022\)](#) explain a phase change in the training of language models by reference to *induction heads*, a family of circuits that produce in-context learning. In concurrent work, [Singh et al.](#) show that the in-context learning from induction heads is later replaced by in-weights learning in the absence of weight decay, but remains strong when weight decay is present. We hypothesise that this effect is also explained through circuit efficiency: the in-context learning from induction heads is a generalising algorithm and so is favoured by weight decay given a large enough dataset size.

[Michaud et al. \(2023\)](#) propose an explanation for power-law scaling ([Barak et al., 2022](#); [Hestness et al., 2017](#); [Kaplan et al., 2020](#)) based on a model in which there are many discrete quanta (algorithms) and larger models learn more of them. Our explanation involves a similar structure: we posit the existence of two algorithms (quanta), and analyse the resulting training dynamics.

## 7. Discussion

**Rethinking generalisation.** [Zhang et al. \(2021\)](#) pose the question of why deep neural networks achieve good generalisation even when they are easily capable of memorising a random labelling of the training data. Our results gesture at a resolution: in the presence of weight decay, circuits that generalise well are likely to be more efficient given a large enough dataset, and thus are preferred over memorisation circuits, even when both achieve perfect training loss (Section 4.1). Similar arguments may hold for other types of regularisation as well.

**Necessity of weight decay.** The biggest limitation of our explanation is that it relies crucially on weight decay, but grokking has been observed even when weight decay is not present (Power et al., 2021; Thilak et al., 2022) (though it is slower and often much harder to elicit (Nanda et al., 2023, Appendix D.1)). This demonstrates that our explanation is incomplete.

Does it also imply that our explanation is *incorrect*? We do not think so. We speculate there is at least one other effect that has a similar regularising effect favouring  $C_{\text{gen}}$  over  $C_{\text{mem}}$ , such as the implicit regularisation of gradient descent (Lyu and Li, 2019; Smith and Le, 2017; Soudry et al., 2018; Wang et al., 2021), and that the speed of the transition from  $C_{\text{mem}}$  to  $C_{\text{gen}}$  is based on the *sum* of these effects and the effect from weight decay. This would neatly explain why grokking takes longer as weight decay decreases (Power et al., 2021), and does not completely vanish in the absence of weight decay. Given that there is a potential extension of our theory that explains grokking without weight decay, and the significant confirming evidence that we have found for our theory in settings with weight decay, we are overall confident that our explanation is at least one part of the true explanation when weight decay is present.

**Broader applicability: beyond parameter norm.** Another limitation is that we only consider one kind of constraint that gradient descent must navigate: parameter norm. Typically, there are many other constraints: fitting the training data, capacity in “bottleneck activations” (Elhage et al., 2021), interference between circuits (Elhage et al., 2022), and more. This may limit the broader applicability of our theory, despite its success in explaining grokking.

**Broader applicability: realistic settings.** A natural question is what implications our theory has for more realistic settings. We expect that the general concepts of circuits, efficiency, and speed of learning continue to apply. However, in realistic settings, good training performance is typically achieved when the model has many different circuit families that contribute different aspects (e.g. language modelling requires spelling, grammar, arithmetic, etc). We expect that these will have a wide variety of learning speeds and efficiencies (although note that “efficiency” is not as well defined in this setting, because the circuits don’t get perfect training accuracy).

In contrast, the key property for grokking in “algorithmic” tasks like modular arithmetic is that there are two clusters of circuit families – one slowly learned, efficient, generalising cluster, and one quickly learned, inefficient, memorising cluster. In particular, our explanation relies on there being no circuits in between the two clusters. Therefore we observe a sharp transition in test performance when shifting from the memorising to the generalising cluster.

**Future work.** Within grokking, several interesting puzzles are still left unexplained. Why does the time taken to grok rise super-exponentially as dataset size decreases? How does the random initialisation interact with efficiency to determine which circuits are found by gradient descent? What causes generalising circuits to develop slower? Investigating these puzzles is a promising avenue for further work.

While the direct application of our work is to understand the puzzle of grokking, we are excited about the potential for understanding deep learning more broadly through the lens of circuit efficiency. We would be excited to see work looking at the role of circuit efficiency in more realistic settings, and work that extends circuit efficiency to consider other constraints that gradient descent must navigate.

## 8. Conclusion

The central question of our paper is: in grokking, why does the network’s test performance improve dramatically upon continued training, having already achieved nearly perfect training performance? Our explanation is: the generalising solution is more “efficient” but slower to learn than the memorising

solution. After quickly learning the memorising circuit, gradient descent can still decrease loss even further by simultaneously strengthening the efficient, generalising circuit and weakening the inefficient, memorising circuit.

Based on our theory we predict and demonstrate two novel behaviours: *ungrokking*, in which a model that has perfect generalisation returns to memorisation when it is further trained on a dataset with size smaller than the critical threshold, and *semi-grokking*, where we train a randomly initialised network on the critical dataset size which results in a grokking-like transition to middling test accuracy. Our explanation is the only one we are aware of that has made (and confirmed) such surprising advance predictions, and we have significant confidence in the explanation as a result.

## Acknowledgements

Thanks to Paul Christiano, Xander Davies, Seb Farquhar, Geoffrey Irving, Tom Lieberum, Eric Michaud, Vlad Mikulik, Neel Nanda, Jonathan Uesato, and anonymous reviewers for valuable discussions and feedback.

## References

- B. Barak, B. Edelman, S. Goel, S. Kakade, E. Malach, and C. Zhang. Hidden progress in deep learning: SGD learns parities near the computational limit. *Advances in Neural Information Processing Systems*, 35:21750–21764, 2022.
- N. Cammarata, G. Goh, S. Carter, C. Voss, L. Schubert, and C. Olah. Curve circuits. *Distill*, 2021. doi: 10.23915/distill.00024.006. <https://distill.pub/2020/circuits/curve-circuits>.
- B. Chughtai, L. Chan, and N. Nanda. A toy model of universality: Reverse engineering how networks learn group operations. *arXiv preprint arXiv:2302.03025*, 2023.
- X. Davies, L. Langosco, and D. Krueger. Unifying grokking and double descent. *arXiv preprint arXiv:2303.06173*, 2023.
- N. Elhage, N. Nanda, C. Olsson, T. Henighan, N. Joseph, B. Mann, A. Askell, Y. Bai, A. Chen, T. Conerly, N. DasSarma, D. Drain, D. Ganguli, Z. Hatfield-Dodds, D. Hernandez, A. Jones, J. Kernion, L. Lovitt, K. Ndousse, D. Amodei, T. Brown, J. Clark, J. Kaplan, S. McCandlish, and C. Olah. A mathematical framework for Transformer circuits. *Transformer Circuits Thread*, 2021. <https://transformer-circuits.pub/2021/framework/index.html>.
- N. Elhage, T. Hume, C. Olsson, N. Schiefer, T. Henighan, S. Kravec, Z. Hatfield-Dodds, R. Lasenby, D. Drain, C. Chen, R. Grosse, S. McCandlish, J. Kaplan, D. Amodei, M. Wattenberg, and C. Olah. Toy models of superposition. *Transformer Circuits Thread*, 2022. [https://transformer-circuits.pub/2022/toy\\_model/index.html](https://transformer-circuits.pub/2022/toy_model/index.html).
- D. Erhan, Y. Bengio, A. Courville, and P. Vincent. Visualizing higher-layer features of a deep network. *University of Montreal*, 1341(3):1, 2009.
- M. Geva, R. Schuster, J. Berant, and O. Levy. Transformer feed-forward layers are key-value memories. *arXiv preprint arXiv:2012.14913*, 2020.
- J. Hestness, S. Narang, N. Ardalani, G. Diamos, H. Jun, H. Kianinejad, M. Patwary, M. Ali, Y. Yang, and Y. Zhou. Deep learning scaling is predictable, empirically. *arXiv preprint arXiv:1712.00409*, 2017.



- A. Jermyn and B. Shlegeris. Multi-component learning and s-curves, 2022. URL <https://www.alignmentforum.org/posts/RKDQCB6smLWgs2Mhr/multi-component-learning-and-s-curves>.
- J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.
- A. Krogh and J. Hertz. A simple weight decay can improve generalization. *Advances in neural information processing systems*, 4, 1991.
- K. Li, A. K. Hopkins, D. Bau, F. Viégas, H. Pfister, and M. Wattenberg. Emergent world representations: Exploring a sequence model trained on a synthetic task. *arXiv preprint arXiv:2210.13382*, 2022.
- Z. Liu, O. Kitouni, N. Nolte, E. J. Michaud, M. Tegmark, and M. Williams. Towards understanding grokking: An effective theory of representation learning. *arXiv preprint arXiv:2205.10343*, 2022.
- Z. Liu, E. J. Michaud, and M. Tegmark. Omnigrok: Grokking beyond algorithmic data, 2023.
- I. Loshchilov and F. Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2019.
- K. Lyu and J. Li. Gradient descent maximizes the margin of homogeneous neural networks. *arXiv preprint arXiv:1906.05890*, 2019.
- M. McCloskey and N. Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. *Psychology of Learning and Motivation*, 24:109–165, 1989.
- K. Meng, D. Bau, A. Andonian, and Y. Belinkov. Locating and editing factual knowledge in GPT. *arXiv preprint arXiv:2202.05262*, 2022.
- W. Merrill, N. Tsilivis, and A. Shukla. A tale of two circuits: Grokking as competition of sparse and dense subnetworks. *arXiv preprint arXiv:2303.11873*, 2023.
- E. J. Michaud, Z. Liu, U. Girit, and M. Tegmark. The quantization model of neural scaling. *arXiv preprint arXiv:2303.13506*, 2023.
- B. Millidge. Grokking ‘grokking’, 2022. URL <https://www.beren.io/2022-01-11-Grokking-Grokking/>.
- N. Nanda, L. Chan, T. Liberman, J. Smith, and J. Steinhardt. Progress measures for grokking via mechanistic interpretability. *arXiv preprint arXiv:2301.05217*, 2023.
- P. Notsawo Jr, H. Zhou, M. Pezeshki, I. Rish, G. Dumas, et al. Predicting grokking long before it happens: A look into the loss landscape of models which grok. *arXiv preprint arXiv:2306.13253*, 2023.
- C. Olah, N. Cammarata, L. Schubert, G. Goh, M. Petrov, and S. Carter. Zoom in: An introduction to circuits. *Distill*, 2020. doi: 10.23915/distill.00024.001. <https://distill.pub/2020/circuits/zoom-in>.
- C. Olsson, N. Elhage, N. Nanda, N. Joseph, N. DasSarma, T. Henighan, B. Mann, A. Askell, Y. Bai, A. Chen, T. Conerly, D. Drain, D. Ganguli, Z. Hatfield-Dodds, D. Hernandez, S. Johnston, A. Jones, J. Kernion, L. Lovitt, K. Ndousse, D. Amodei, T. Brown, J. Clark, J. Kaplan, S. McCandlish, and C. Olah. In-context learning and induction heads. *Transformer Circuits Thread*, 2022. <https://transformer-circuits.pub/2022/in-context-learning-and-induction-heads/index.html>.
-

- A. Power, Y. Burda, H. Edwards, I. Babuschkin, and V. Misra. Grokking: Generalization beyond overfitting on small algorithmic datasets. In *Mathematical Reasoning in General Artificial Intelligence Workshop, ICLR*, 2021. URL [https://mathai-iclr.github.io/papers/papers/MATHAI\\_29\\_paper.pdf](https://mathai-iclr.github.io/papers/papers/MATHAI_29_paper.pdf).
- R. Ratcliff. Connectionist models of recognition memory: constraints imposed by learning and forgetting functions. *Psychology Review*, 97(2):285–308, 1990.
- A. Singh, S. Chan, T. Moskovitz, E. Grant, A. Saxe, and F. Hill. The transient nature of emergent in-context learning in transformers. *Forthcoming*.
- S. L. Smith and Q. V. Le. A bayesian perspective on generalization and stochastic gradient descent. *arXiv preprint arXiv:1710.06451*, 2017.
- D. Soudry, E. Hoffer, M. S. Nacson, S. Gunasekar, and N. Srebro. The implicit bias of gradient descent on separable data. *The Journal of Machine Learning Research*, 19(1):2822–2878, 2018.
- V. Thilak, E. Littwin, S. Zhai, O. Saremi, R. Paiss, and J. Susskind. The slingshot mechanism: An empirical study of adaptive optimizers and the grokking phenomenon. *arXiv preprint arXiv:2206.04817*, 2022.
- A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, 2017. URL [https://proceedings.neurips.cc/paper\\_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf](https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf).
- B. Wang, Q. Meng, W. Chen, and T.-Y. Liu. The implicit bias for adaptive optimization algorithms on homogeneous neural networks. In *International Conference on Machine Learning*, pages 10849–10858. PMLR, 2021.
- K. Wang, A. Variengien, A. Conmy, B. Shlegeris, and J. Steinhardt. Interpretability in the wild: a circuit for indirect object identification in gpt-2 small. *arXiv preprint arXiv:2211.00593*, 2022.
- C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals. Understanding deep learning (still) requires rethinking generalization. *Communications of the ACM*, 64(3):107–115, 2021.
- G. Zhang, C. Wang, B. Xu, and R. Grosse. Three mechanisms of weight decay regularization. *arXiv preprint arXiv:1810.12281*, 2018.

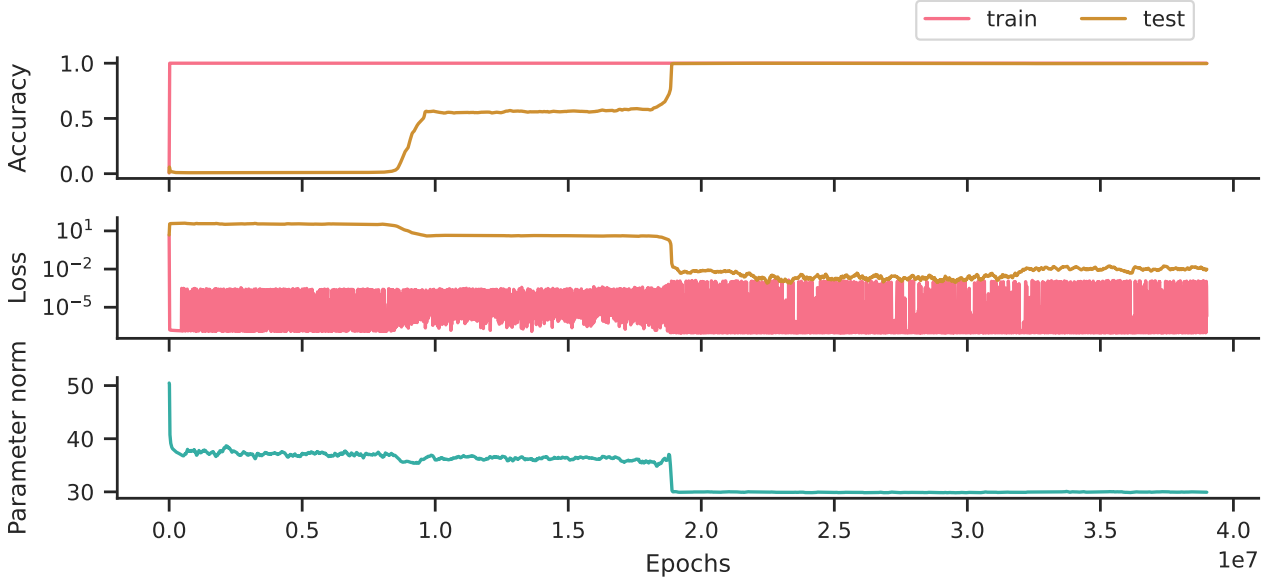


Figure 6 | **Examining a single semi-grokking run in detail.** We plot accuracy, loss, and parameter norm over training for a single cherry-picked modular addition run at a dataset size of 1532 (12% of the full dataset). This run shows transient semi-grokking. At epoch  $0.8 \times 10^7$ , test accuracy rises to around 0.55, and then stays there for  $10^7$  epochs, because  $C_{\text{gen}}$  and  $C_{\text{mem}}$  efficiencies are balanced. At epoch  $1.8 \times 10^7$ , we speculate that gradient descent finds an even more efficient  $C_{\text{gen}}$  circuit, as parameter norm drops suddenly and test accuracy rises to 1. At epoch  $3.2 \times 10^7$  we see test loss rise, we do not know why. There seem to be multiple phases, perhaps corresponding to the network transitioning between mixtures of multiple circuits with increasing efficiencies, but further investigation is needed.

## A. Experimental details and more evidence

For all our experiments, we use 1-layer decoder-only transformer networks (Vaswani et al., 2017) with learned positional embeddings, untied embeddings/unembeddings. The hyperparameters are as follows:  $d_{\text{model}} = 128$  is the residual stream width,  $d_{\text{head}} = 32$  is the size of the query, key, and value vectors for each attention head,  $d_{\text{mlp}} = 512$  is the number of neurons in the hidden layer of the MLP, and we have  $d_{\text{model}}/d_{\text{head}} = 4$  heads per self-attention layer. We optimise the network with full batch training (that is, using the entire training dataset for each update) using the AdamW optimiser (Loshchilov and Hutter, 2019) with  $\beta_1 = 0.9$ ,  $\beta_2 = 0.98$ , learning rate of  $10^{-3}$ , and weight decay of 1.0. In some of our experiments we vary the weight decay in order to produce networks with varying parameter norm.

Following Power et al. (2021), for a binary operation  $x \circ y$ , we construct a dataset of the form  $\langle x \rangle \langle \circ \rangle \langle y \rangle \langle = \rangle \langle x \circ y \rangle$ , where  $\langle a \rangle$  stands for the token corresponding to the element  $a$ . We choose a fraction of this dataset at random as the train dataset, and the remainder as the test dataset. The first 4 tokens  $\langle x \rangle \langle \circ \rangle \langle y \rangle \langle = \rangle$  are the input to the network, and we train with cross-entropy loss over the final token  $\langle x \circ y \rangle$ . For all modular arithmetic tasks we use the modulus  $p = 113$ , so for example the size of the full dataset for modular addition is  $p^2 = 12769$ , and  $d_{\text{vocab}} = 115$ , including the  $\langle + \rangle$  and  $\langle = \rangle$  tokens.

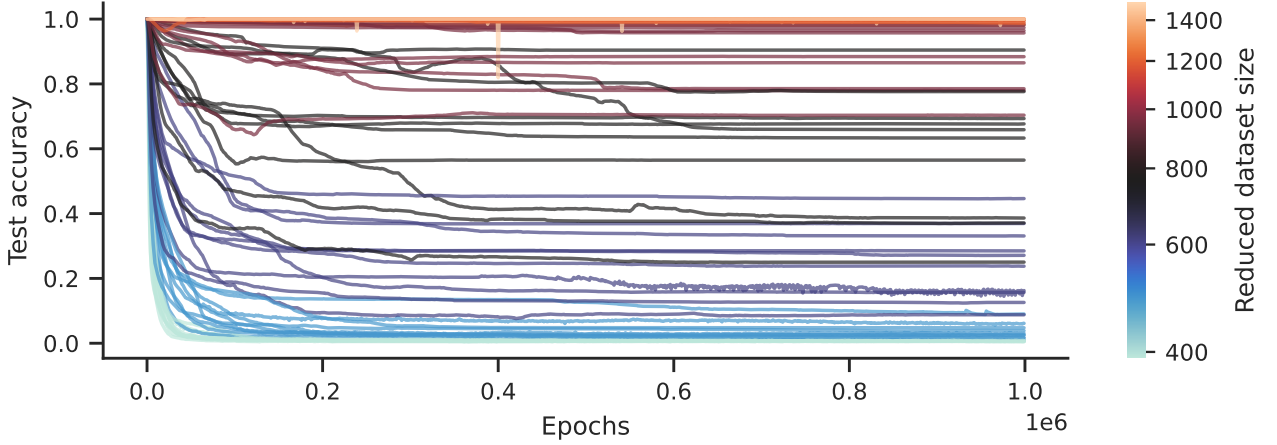


Figure 7 | **Many ungrokking runs.** We show test accuracy over epochs for a range of ungrokking runs for modular addition. Each line represents a single run, and we sweep over 7 geometrically spaced dataset sizes in  $[390, 1494]$  with 10 seeds each. Each run is initialised with parameters from a network trained on the full dataset (the initialisation runs are not shown), so test accuracy starts at 1 for all runs. When the dataset size is small enough, the network ungroks to poor test accuracy, while train accuracy remains at 1 (not shown). For an intermediate dataset size, we see ungrokking to middling test accuracy as  $C_{\text{gen}}$  and  $C_{\text{mem}}$  efficiencies are balanced.

### A.1. Semi-grokking

In Section 5.3 we looked at all semi-grokking training runs in Figure 5. Here, we investigate a single example of transient semi-grokking in more detail (see Figure 6). We speculate that there are multiple circuits with increasing efficiencies for  $C_{\text{gen}}$ , and in these cases the more efficient circuits are slower to learn. This would explain transient semi-grokking: gradient descent first finds a less efficient  $C_{\text{gen}}$  and we see middling generalisation, but since we are using the upper range of  $D_{\text{crit}}$ , eventually gradient descent finds a more efficient  $C_{\text{gen}}$  leading to full generalisation.

### A.2. Ungrokking

In Figure 7, we show many ungrokking runs for modular addition, and in Figure 8 we show ungrokking across many other tasks.

We have already seen that  $D_{\text{crit}}$  is affected by the random initialisation. It is interesting to compare  $D_{\text{crit}}$  when starting with a given random initialisation, and when ungrokking from a network that was trained to full generalisation with the same random initialisation. Figure 5 shows a semi-grokking run that achieves a test accuracy of  $\sim 0.7$  with a dataset size of  $\sim 2000$ , while Figure 7 shows ungrokking runs that achieve a test accuracy of  $\sim 0.7$  with a dataset size of around 800–1000, less than half of what the semi-grokking run required.

In Figure 10b, the final test accuracy after *ungrokking* shows a smooth relationship with dataset size, which we might expect if  $C_{\text{gen}}$  is getting stronger on a smoothly increasing number of inputs compared to  $C_{\text{mem}}$ . However due to the difficulties discussed previously, we don't see a smooth relationship between test accuracy and dataset size in *semi-grokking*.

These results suggest that  $D_{\text{crit}}$  is an oversimplified concept, because in reality the initialisation and training dynamics affect which circuits are found, and therefore the dataset size at which we see middling generalisation.

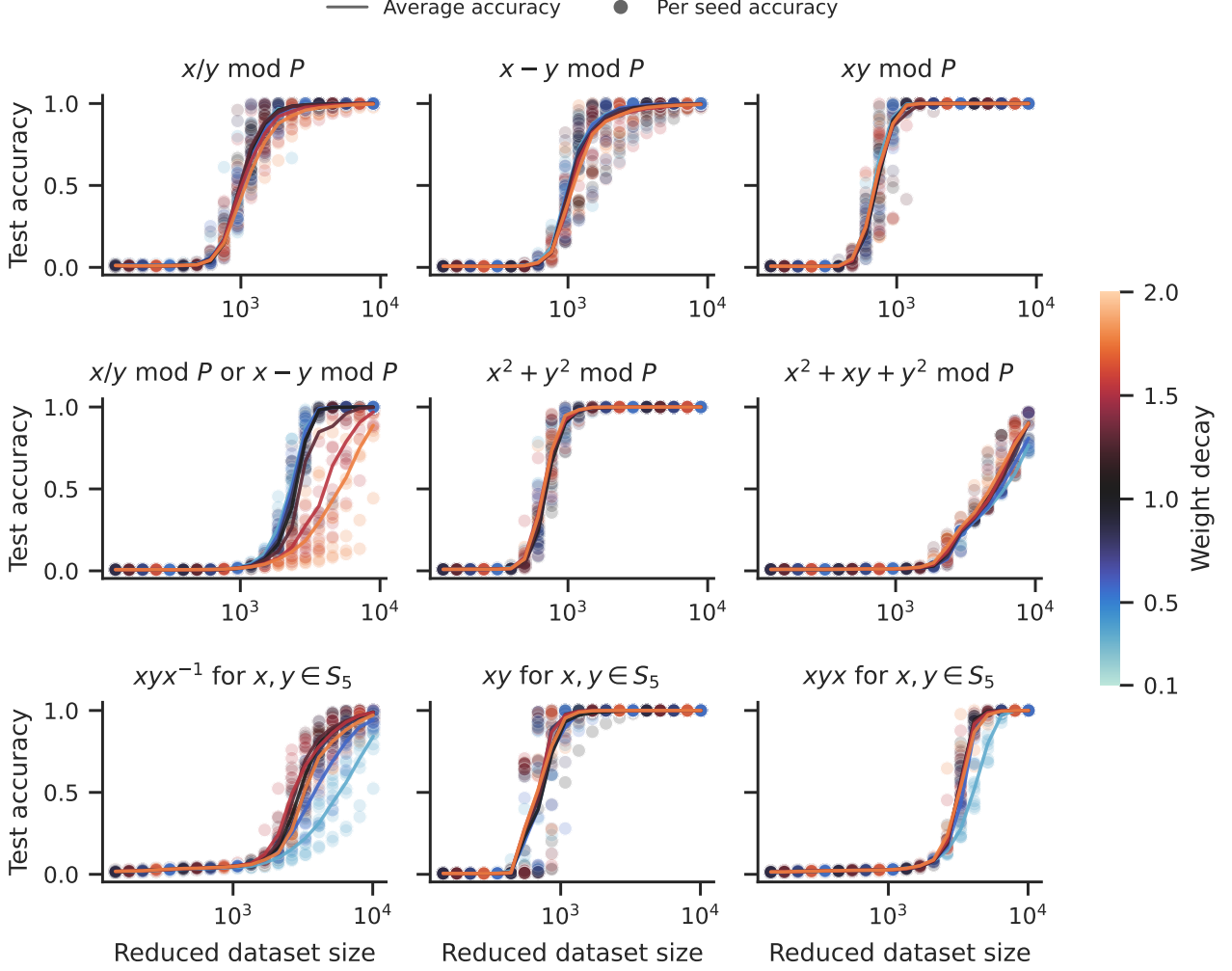


Figure 8 | **Ungrokking on many other tasks.** We plot test accuracy against reduced dataset size for many other modular arithmetic and symmetric group tasks (Power et al., 2021). For each run, we train on the full dataset (achieving 100% accuracy), and then further train on a reduced subset of the dataset for 100k steps. The results show clear ungrokking, since in many cases test accuracy falls below 100%, often to nearly 0%. For most datasets the transition point is independent of weight decay (different coloured lines almost perfectly overlap).



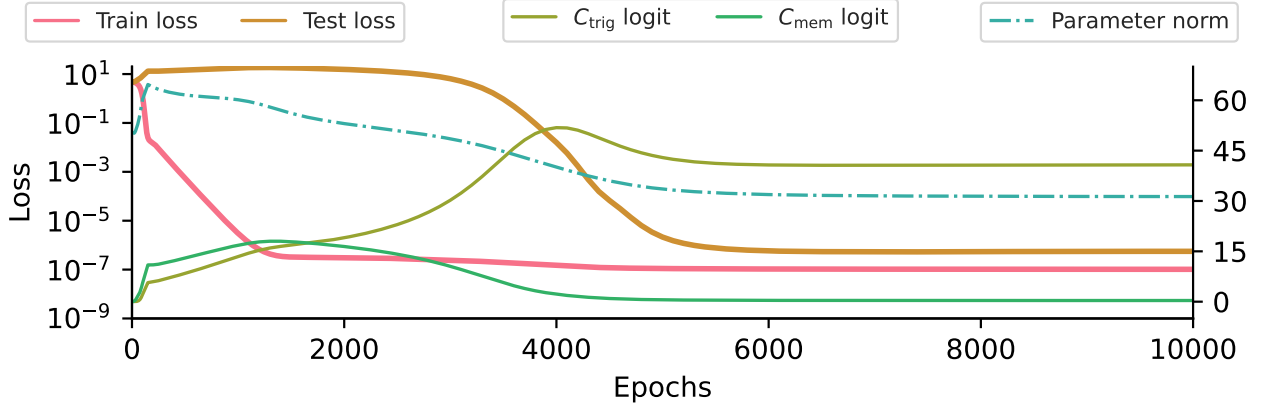


Figure 9 | **Grokking occurs because  $C_{\text{gen}}$  is more efficient than  $C_{\text{mem}}$ .** We show loss, parameter norm, and the value of the correct logit for  $C_{\text{gen}}$  and  $C_{\text{mem}}$  for a randomly-picked training run. By step 200, the train accuracy is already perfect (not shown), train loss is low while test loss has risen, and parameter norm is at its maximum value, indicating strong  $C_{\text{mem}}$ . Train loss continues to fall rapidly until step 1500, as parameter norm falls and the  $C_{\text{gen}}$  logit becomes higher than the  $C_{\text{mem}}$  logit. At step 3500, test loss starts to fall as the high  $C_{\text{gen}}$  logit starts to dominate, and by step 6000 we get good generalisation.

### A.3. $C_{\text{gen}}$ and $C_{\text{mem}}$ development during grokking

In Figure 9 we show  $C_{\text{gen}}$  and  $C_{\text{mem}}$  development via the proxy measures defined in Appendix B for a randomly-picked grokking run. Looking at these measures was very useful to form a working theory for why grokking happens. However as we note in Appendix B, these proxy measures tend to overestimate  $C_{\text{gen}}$  and underestimate  $C_{\text{mem}}$ .

We note some interesting phenomena in Figure 9:

1. Between epochs 200 to 1500, *both* the  $C_{\text{gen}}$  and  $C_{\text{mem}}$  logits are rising while parameter norm is falling, indicating that gradient descent is improving efficiency (possibly by removing irrelevant parameters).
2. After epoch 4000, the  $C_{\text{gen}}$  logit *falls* while the  $C_{\text{mem}}$  logit is already  $\sim 0$ . Since test loss continues to fall, we expect that incorrect logits from  $C_{\text{mem}}$  on the test dataset are getting cleaned up, as described in Nanda et al. (2023).

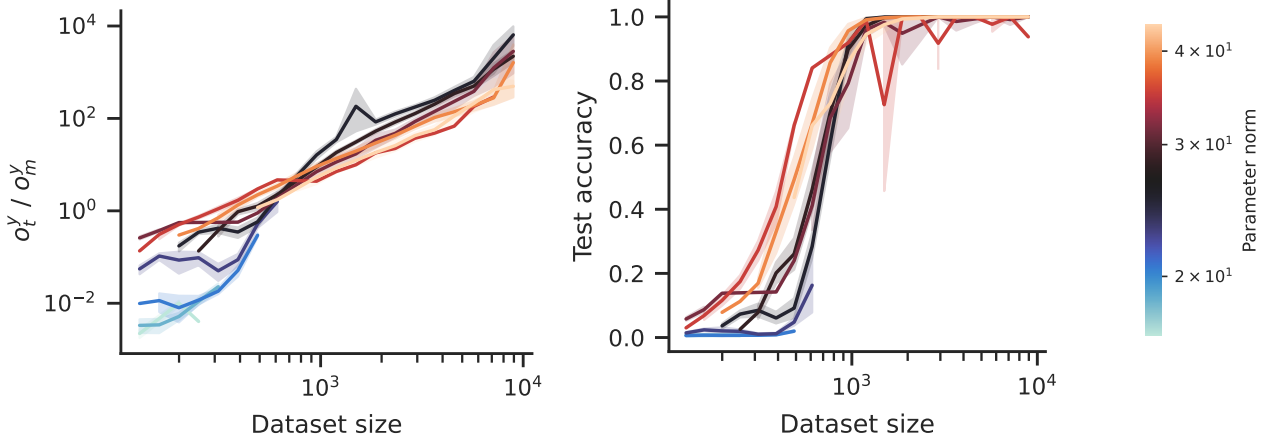
### A.4. Tradeoffs between $C_{\text{gen}}$ and $C_{\text{mem}}$

In Section 4.1 we looked at the efficiency of  $C_{\text{gen}}$ -only and  $C_{\text{mem}}$ -only circuits. In this section we train on varying dataset sizes so that the network develops a mixture of  $C_{\text{gen}}$  and  $C_{\text{mem}}$  circuits, and study their relative strength using the correct logit as a proxy measure (described in Appendix B).

As we demonstrated previously,  $C_{\text{mem}}$ 's efficiency drops with increasing dataset size, while  $C_{\text{gen}}$ 's stays constant. Theorem D.4 (case 2) suggests that parameter norm allocated to a circuit is proportional to efficiency, and since logit values also increase with parameter norm, this implies that the ratio of the  $C_{\text{gen}}$  to  $C_{\text{mem}}$  logit  $o_t^y / o_m^y$  should increase monotonically with dataset size.

In Figure 10a we see exactly this: the logit ratio changes monotonically (over 6 orders of magnitude) with increasing dataset size.

Due to the difficulties in training to convergence at small dataset sizes, we initialised all parameters



(a) **Logit ratio ( $o_t^y / o_m^y$ ) vs dataset size ( $D$ ).** Colours correspond to different bucketed values of parameter norm ( $P$ ). Each line shows that as dataset size increases, a fixed parameter norm (fixed colour) is being reallocated smoothly towards increasing the trigonometric logit compared to the memorisation logit.

(b) **Test accuracy vs dataset size ( $D$ ).** We see a smooth dependence on dataset size. Each line shows that as dataset size increases, the reallocation of a fixed parameter norm (fixed colour) towards  $C_{\text{gen}}$  from  $C_{\text{mem}}$  results in increasing accuracy.

**Figure 10 | Relative strength at convergence.** We report logit ratios and test accuracy at convergence across a range of training runs, generated by sweeping over the weight decay and random seed to obtain different parameter norms at the same dataset size. We use the ungrokking runs from Figure 7, so every run is initialised with parameters obtained by training on the full dataset.

from a  $C_{\text{gen}}$ -only network trained on the full dataset. We confirmed that in all the runs, at convergence, the training loss from the  $C_{\text{gen}}$ -initialised network was lower than the training loss from a randomly initialised network, indicating that this initialisation allows our optimiser to find a better minimum than from random initialisation.

## B. $C_{\text{gen}}$ and $C_{\text{mem}}$ in modular addition

In modular addition, given two integers  $a, b$  and a modulus  $p$  as input, where  $0 \leq a, b < p$ , the task is to predict  $a + b \bmod p$ . Nanda et al. (2023) identified the generalising algorithm implemented by a 1-layer transformer after grokking (visualised in Figure 11), which we call the “trigonometric” algorithm. In this section we summarise the algorithm, and explain how we produce our proxy metrics for the strength of  $C_{\text{gen}}$  and  $C_{\text{mem}}$ .

**Trigonometric logits.** We explain the structure of the logits produced by the trigonometric algorithm. For each possible label  $c \in \{0, 1, \dots, p-1\}$ , the trigonometric logit  $o^c$  will be given by  $\sum_{\omega_k} \cos(\omega_k(a + b - c))$ , for a few key frequencies  $\omega_k = 2\pi \frac{k}{p}$  with integer  $k$ . For the true label  $c^* = a + b \bmod p$ , the term  $\omega_k(a + b - c^*)$  is an integer multiple of  $2\pi$ , and so  $\cos(\omega_k(a + b - c^*)) = 1$ . For any incorrect label  $c \neq a + b \bmod p$ , it is very likely that at least some of the key frequencies satisfy  $\cos(\omega_k(a + b - c)) \ll 1$ , creating a large difference between  $o^c$  and  $o^{c^*}$ .

**Trigonometric algorithm.** There is a set of key frequencies  $\omega_k$ . (These frequencies are typically whichever frequencies were highest at the time of random initialisation.) For an arbitrary label  $c$ , the logit  $o^c$  is computed as follows:

1. Embed the one-hot encoded number  $a$  to  $\sin(\omega_k a)$  and  $\cos(\omega_k a)$  for the various frequencies  $\omega_k$ .

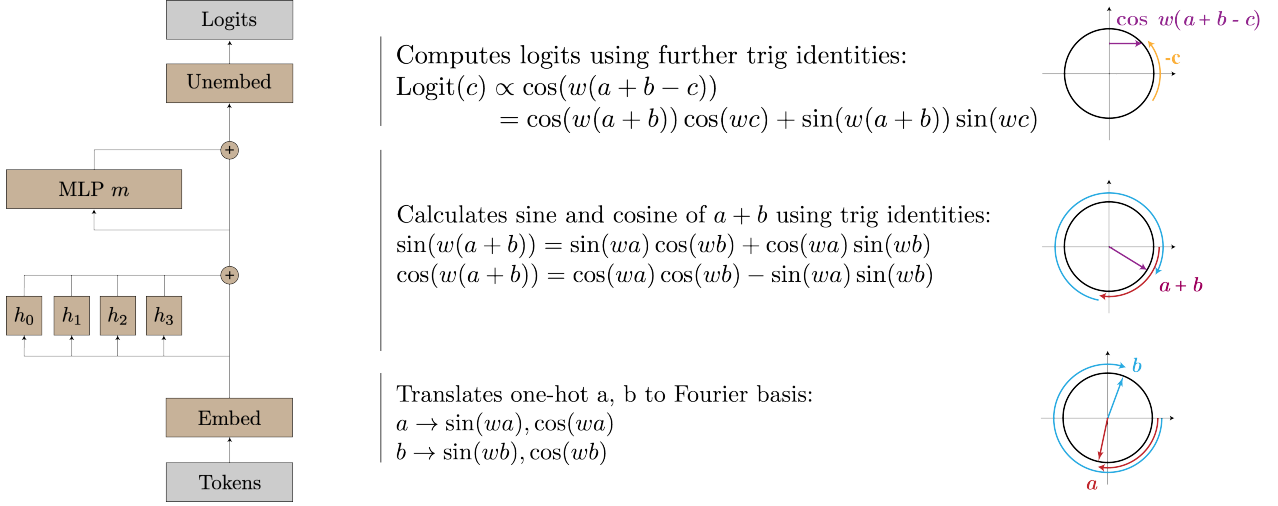


Figure 11 | **The trigonometric algorithm for modular arithmetic** (reproduced from [Nanda et al. \(2023\)](#)). Given two numbers  $a$  and  $b$ , the model projects each point onto a corresponding rotation using its embedding matrix. Using its attention and MLP layers, it then composes the rotations to get a representation of  $a+b \bmod p$ . Finally, it “reads off” the logits for each  $c \in \{0, 1, \dots, p-1\}$ , by rotating by  $-c$  to get  $\cos(\omega(a+b-c))$ , which is maximised when  $a+b \equiv c \bmod P$  (since  $\omega$  is a multiple of  $2\pi$ ).

Do the same for  $b$ .

2. Compute  $\cos(\omega_k(a+b))$  and  $\sin(\omega_k(a+b))$  using the intermediate attention and MLP layers via the trigonometric identities:

$$\begin{aligned}\cos(\omega_k(a+b)) &= \cos(\omega_k a) \cos(\omega_k b) - \sin(\omega_k a) \sin(\omega_k b) \\ \sin(\omega_k(a+b)) &= \sin(\omega_k a) \cos(\omega_k b) + \cos(\omega_k a) \sin(\omega_k b)\end{aligned}$$

3. Use the output and unembedding matrices to implement the trigonometric identity:

$$o^c = \sum_{\omega_k} \cos(\omega_k(a+b-c)) = \sum_{\omega_k} \cos(\omega_k(a+b)) \cos(\omega_k c) + \sin(\omega_k(a+b)) \sin(\omega_k c).$$

**Isolating trigonometric logits.** Given a classifier  $h$ , we can aggregate its logits on every possible input, resulting in a vector  $\vec{Z}_h$  of length  $p^3$  where  $\vec{Z}_h^{a,b,c} = o_h^c$  (“ $a+b=$ ”) is the logit for label  $c$  on the input  $(a,b)$ . We are interested in identifying the contribution of the trigonometric algorithm to  $\vec{Z}_h$ . We use the same method as [Chughtai et al. \(2023\)](#) and restrict  $\vec{Z}_h$  to a much smaller trigonometric subspace.

For a frequency  $\omega_k$ , let us define the  $p^3$ -dimensional vector  $\vec{Z}_{\omega_k}$  as  $\vec{Z}_{\omega_k}^{a,b,c} = \cos(\omega_k(a+b-c))$ . Since  $\vec{Z}_{\omega_k} = \vec{Z}_{\omega_{p-k}}$ , we set  $1 \leq k \leq K$ , where  $K = \lceil (p-1)/2 \rceil$ , to obtain  $K$  distinct vectors, ignoring the constant bias vector. These vectors are orthogonal, as they are part of a Fourier basis.

Notice that any circuit that was exactly following the learned algorithm described above would only produce logits in the directions  $\vec{Z}_{\omega_k}$  for the key frequencies  $\omega_k$ . So, we can define the trigonometric contribution to  $\vec{Z}_h$  as the projection of  $\vec{Z}_h$  onto the directions  $\vec{Z}_{\omega_k}$ . We may not know the key frequencies in advance, but we can sum over all  $K$  of them, giving the following definition for trigonometric logits:

$$\vec{Z}_{h,T} = \sum_{k=1}^K (\vec{Z}_h \cdot \hat{Z}_{\omega_k}) \hat{Z}_{\omega_k}$$

where  $\hat{Z}_{\omega_k}$  is the normalised version of  $\vec{Z}_{\omega_k}$ . This corresponds to projecting onto a  $K$ -dimensional subspace of the  $p^3$ -dimensional space in which  $\vec{Z}_h$  lives.

**Memorisation logits.** Early in training, neural networks memorise the training dataset without generalising, suggesting that there exists a memorisation algorithm, implemented by the circuit  $C_{\text{mem}}$ <sup>1</sup>. Unfortunately, we do not understand the algorithm underlying memorisation, and so cannot design a similar procedure to isolate  $C_{\text{mem}}$ 's contribution to the logits. However, we hypothesise that for modular addition,  $C_{\text{gen}}$  and  $C_{\text{mem}}$  are the only two circuit families of importance for the loss. This allows us to define the  $C_{\text{mem}}$  contribution to the logits as the residual:

$$\vec{Z}_{h,M} = \vec{Z}_h - \vec{Z}_{h,T}$$

**$C_{\text{trig}}$  and  $C_{\text{mem}}$  circuits.** We say that a circuit is a  $C_{\text{trig}}$  circuit if it implements the  $C_{\text{trig}}$  algorithm, and similarly for  $C_{\text{mem}}$  circuits. Importantly, this is a many-to-one mapping: there are many possible circuits that implement a given algorithm.

We isolate  $C_{\text{trig}}$  ( $\vec{o}_t$ ) and  $C_{\text{mem}}$  ( $\vec{o}_m$ ) logits by projecting the output logits ( $\vec{o}$ ) as described in Appendix B. We cannot directly measure the circuit weights  $w_t$  and  $w_m$ , but instead use an indirect measure: the value of the logit for the correct class given by each circuit, i.e.  $o_t^y$  and  $o_m^y$ .

**Flaws** These metrics should be viewed as an imperfect proxy measure for the true strength of the trigonometric and memorisation circuits, as they have a number of flaws:

1. When both  $C_{\text{trig}}$  and  $C_{\text{mem}}$  are present in the network, they are both expected to produce high values for the correct logits, and low values for incorrect logits, on the train dataset. Since the  $C_{\text{trig}}$  and  $C_{\text{mem}}$  logits are correlated, it becomes more likely that  $\vec{Z}_{h,T}$  captures  $C_{\text{mem}}$  logits too.
2. In this case we would expect our proxy measure to overestimate the strength of  $C_{\text{trig}}$  and underestimate the strength of  $C_{\text{mem}}$ . In fact, in our experiments we do see large negative correct logit values for  $C_{\text{mem}}$  on training for semi-grokking, which probably arises because of this effect.
3. Logits are not inherently meaningful; what matters for loss is the extent to which the correct logit is larger than the incorrect logits. This is not captured by our proxy metric, which only looks at the size of the correct logit. In a binary classification setting, we could instead use the difference between the correct and incorrect logit, but it is not clear what a better metric would be in the multiclass setting.

## C. Details for the minimal example

In Figure 2 we show that two ingredients: multiple circuits with different efficiencies, and slow and fast circuit development, are sufficient to reproduce learning curves that qualitatively demonstrate grokking. In Table 1 we provide details about the simulation used to produce this figure.

As explained in Section 3, the logits produced by  $C_{\text{gen}}$  and  $C_{\text{mem}}$  are given by:

$$o_G^y(x) = \mathbb{1} [(x, y) \in \mathcal{D} \text{ or } (x, y) \in \mathcal{D}_{\text{test}}] \quad (3)$$

$$o_M^y(x) = \mathbb{1} [(x, y) \in \mathcal{D} \text{ or } (x, y) \in \mathcal{D}_{\text{mem}}] \quad (4)$$

<sup>1</sup>In reality, there are at least two different memorisation algorithms: commutative memorisation (which predicts the same answer for  $(a, b)$  and  $(b, a)$ ) and non-commutative memorisation (which does not). However, this difference does not matter for our analyses, and we will call both of these ‘‘memorisation’’ in this paper.

Table 1 | Hyperparameters used for our simulations.

(a)  $C_{\text{gen}}$  learned slower but more efficient than  $C_{\text{mem}}$ .

Parameter	Value
$P_g$	1
$P_m$	2
$\kappa$	1.2
$\alpha$	0.005
$w_{g_1}(0)$	0
$w_{g_2}(0)$	0.005
$w_{m_1}(0)$	0
$w_{m_2}(0)$	1
$q$	113
$\lambda$	0.01

(b)  $C_{\text{gen}}$  less efficient than  $C_{\text{mem}}$ .

Parameter	Value
$P_g$	4
$P_m$	2
$\kappa$	1.2
$\alpha$	0.005
$w_{g_1}(0)$	0
$w_{g_2}(0)$	0.005
$w_{m_1}(0)$	0
$w_{m_2}(0)$	1
$q$	113
$\lambda$	0.01

(c)  $C_{\text{gen}}$  and  $C_{\text{mem}}$  learned at equal speeds.

Parameter	Value
$P_g$	1
$P_m$	2
$\kappa$	1.2
$\alpha$	0.005
$w_{g_1}(0)$	0
$w_{g_2}(0)$	1
$w_{m_1}(0)$	0
$w_{m_2}(0)$	1
$q$	113
$\lambda$	0.01

These are scaled by two independent weights for each circuit, giving the overall logits as:

$$o^y(x) = w_{G_1} w_{G_2} o_G^y(x) + w_{M_1} w_{M_2} o_M^y(x) \quad (5)$$

We model the parameter norms according to the scaling efficiency in Section D.1, inspired by a  $\kappa$ -layer MLP with Relu activations and without biases:

$$P'_c = (w_{c_1} w_{c_2})^{1/\kappa} P_c \text{ for } c \in (g, m).$$

From Equations (3) to (5) we get the following equations for train and test loss respectively:

$$\begin{aligned} \mathcal{L}_{\text{train}} &= -\log \frac{\exp(w_{g_1} w_{g_2} + w_{m_1} w_{m_2})}{(q-1) + \exp(w_{g_1} w_{g_2} + w_{m_1} w_{m_2})} + \mathcal{L}_{\text{wd}}, \\ \mathcal{L}_{\text{test}} &= -\log \frac{\exp(w_{g_1} w_{g_2})}{(q-2) + \exp(w_{g_1} w_{g_2}) + \exp(w_{m_1} w_{m_2})} + \mathcal{L}_{\text{wd}}, \end{aligned}$$

where  $q$  is the number of labels, and the weight decay loss is:

$$\mathcal{L}_{\text{wd}} = P_g'^2 + P_m'^2.$$

The weights  $w_{c_i}$  are updated based on gradient descent:

$$w_{c_i}(\tau) \leftarrow w_{c_i}(\tau-1) - \lambda \frac{\partial \mathcal{L}_{\text{train}}}{\partial w_{c_i}}$$

where  $\lambda$  is a learning rate. The initial values of the parameters are  $w_{c_i}(0)$ . In Table 1 we list the values of the simulation hyperparameters.

## D. Proofs of theorems

We assume we have a set of inputs  $X$ , a set of labels  $Y$ , and a training dataset,  $\mathcal{D} = \{(x_1, y_1), \dots (x_D, y_D)\}$ . Let  $h$  be a classifier that assigns a real-valued logit for each possible label given an input. We denote



an individual logit as  $o_h^y(x) := h(x, y)$ . When the input  $x$  is clear from context, we will denote the logit as  $o_h^y$ . Excluding weight decay, the loss for the classifier is given by the softmax cross-entropy loss:

$$\mathcal{L}_{\text{x-ent}}(h) = -\frac{1}{D} \sum_{(x,y) \in \mathcal{D}} \log \frac{\exp(o_h^y)}{\sum_{y' \in Y} \exp(o_h^{y'})}.$$

For any  $c \in \mathbb{R}$ , let  $c \cdot h$  be the classifier whose logits are multiplied by  $c$ , that is,  $(c \cdot h)(x, y) = c \times h(x, y)$ . Intuitively, once a classifier achieves perfect accuracy, then the true class logit  $o^{y^*}$  will be larger than any incorrect class logit  $o^{y'}$ , and so loss can be further reduced by scaling up *all* of the logits further (increasing the gap between  $o^{y^*}$  and  $o^{y'}$ ).

**Theorem D.1.** *Suppose that the classifier  $h$  has perfect accuracy, that is, for any  $(x, y^*) \in \mathcal{D}$  and any  $y' \neq y^*$  we have  $o_h^{y^*} > o_h^{y'}$ . Then, for any  $c > 1$ , we have  $\mathcal{L}_{\text{x-ent}}(c \cdot h) < \mathcal{L}_{\text{x-ent}}(h)$ .*

*Proof.* First, note that we can rewrite the loss function as:

$$\mathcal{L}_{\text{x-ent}}(h) = -\frac{1}{D} \sum_{(x,y^*)} \log \frac{\exp(o_h^{y^*})}{\sum_{y'} \exp(o_h^{y'})} = \frac{1}{D} \sum_{(x,y^*)} \log \left( \frac{\sum_{y'} \exp(o_h^{y'})}{\exp(o_h^{y^*})} \right) = \frac{1}{D} \sum_{(x,y^*)} \log \left( 1 + \sum_{y' \neq y^*} \exp(o_h^{y'} - o_h^{y^*}) \right)$$

Since we are given that  $o_h^{y^*} > o_h^{y'}$ , for any  $c > 1$  we have  $c(o_h^{y'} - o_h^{y^*}) < o_h^{y'} - o_h^{y^*}$ . Since  $\exp$ ,  $\log$ , and sums are all monotonic, this gives us our desired result:

$$\mathcal{L}_{\text{x-ent}}(c \cdot h) = \frac{1}{D} \sum_{(x,y^*)} \log \left( 1 + \sum_{y' \neq y^*} \exp(c(o_h^{y'} - o_h^{y^*})) \right) < \frac{1}{D} \sum_{(x,y^*)} \log \left( 1 + \sum_{y' \neq y^*} \exp(o_h^{y'} - o_h^{y^*}) \right) = \mathcal{L}_{\text{x-ent}}(h).$$

□

We now move on to Theorem D.4. First we establish some basic lemmas that will be used in the proof:

**Lemma D.2.** *Let  $a, b, r \in \mathbb{R}$  with  $a, b \geq 0$  and  $0 < r \leq 1$ . Then  $(a + b)^r \leq a^r + b^r$ .*

*Proof.* The case with  $a = 0$  or  $b = 0$  is clear, so let us consider  $a, b > 0$ . Let  $x = \frac{a}{a+b}$  and  $y = \frac{b}{a+b}$ . Since  $0 \leq x \leq 1$ , we have  $x^{(1-r)} \leq 1$ , which implies  $x \leq x^r$ . Similarly  $y \leq y^r$ . Thus  $x^r + y^r \geq x + y = 1$ . Substituting in the values of  $x$  and  $y$  we get  $\frac{a^r + b^r}{(a+b)^r} \geq 1$ , which when rearranged gives us the desired result. □

**Lemma D.3.** *For any  $x, c, r \in \mathbb{R}$  with  $r \geq 1$ , there exists some  $\delta > 0$  such that for any  $\epsilon < \delta$  we have  $x^r - (x - \epsilon)^r > \delta(rx^{r-1} - c)$ .*

*Proof.* The function  $f(x) = x^r$  is everywhere-differentiable and has derivative  $rx^{r-1}$ . Thus we can choose  $\delta$  such that for any  $\epsilon < \delta$  we have  $-c < \frac{x^r - (x-\epsilon)^r}{\epsilon} - rx^{r-1} < c$ . Rearranging, we get  $x^r - (x - \epsilon)^r > \delta(rx^{r-1} - c)$  as desired. □

### D.1. Weight decay favours efficient circuits

To flesh out the argument in Section 3, we construct a minimal example of multiple circuits  $\{C_1, \dots, C_I\}$  of varying efficiencies that can be scaled up or down through a set of non-negative *weights*  $w_i$ . Our classifier is given by  $h = \sum_{i=1}^I w_i C_i$ , that is, the output  $h(x, y)$  is given by  $\sum_{i=1}^I w_i C_i(x, y)$ .

We take circuits  $C_i$  that are *normalised*, that is, they produce the same average logit value.  $P_i$  denotes the parameter norm of the normalised circuit  $C_i$ . We decide to call a circuit with lower  $P_i$  more efficient. However, it is hard to define efficiency precisely. Consider instead the parameter norm  $P'_i$  of the scaled circuit  $w_i C_i$ . If we define efficiency as either the ratio  $\|\vec{o}_{C_i}\|/P'_i$  or the derivative  $d\|\vec{o}_{C_i}\|/dP'_i$ , then it would vary with  $w_i$  since  $\vec{o}_{C_i}$  and  $P'_i$  can in general have different relationships with  $w_i$ . We prefer  $P_i$  as a measure of relative efficiency as it is intrinsic to  $C_i$  rather than depending on its scaling  $w_i$ .

Gradient descent operates over the weights  $w_i$  (but not  $C_i$  or  $P_i$ ) to minimise  $\mathcal{L} = \mathcal{L}_{\text{x-ent}} + \alpha \mathcal{L}_{\text{wd}}$ .  $\mathcal{L}_{\text{x-ent}}$  can easily be rewritten in terms of  $w_i$ , but for  $\mathcal{L}_{\text{wd}}$  we need to model the parameter norm of the scaled circuits  $w_i C_i$ . Notice that, in a  $\kappa$ -layer MLP with Relu activations and without biases, scaling all parameters by a constant  $c$  scales the outputs by  $c^\kappa$ . Inspired by this observation, we model the parameter norm of  $w_i C_i$  as  $w_i^{1/\kappa} P_i$  for some  $\kappa > 0$ . This gives the following effective loss:

$$\mathcal{L}(\vec{w}) = \mathcal{L}_{\text{x-ent}} \left( \sum_{i=1}^I w_i C_i \right) + \frac{\alpha}{2} \sum_{i=1}^I (w_i^{1/\kappa} P_i)^2$$

We will generalise this to any  $L_q$ -norm (where  $q > 0$ ). Standard weight decay corresponds to  $q = 2$ . We will also generalise to arbitrary differentiable, bounded training loss functions, instead of cross-entropy loss specifically. In particular, we assume that there is some differentiable  $\mathcal{L}_{\text{train}}(h)$  such that there exists a finite bound  $B \in \mathbb{R}$  such that  $\forall h : \mathcal{L}_{\text{train}}(h) \geq B$ . (In the case of cross-entropy loss,  $B = 0$ .)

With these generalisations, the overall loss is now given by:

$$\mathcal{L}(\vec{w}) = \mathcal{L}_{\text{train}} \left( \sum_{i=1}^I w_i C_i \right) + \frac{\alpha}{q} \sum_{i=1}^I (w_i^{1/\kappa} P_i)^q \quad (6)$$

The following theorem establishes that the optimal weight vector allocates more weight to more efficient circuits, under the assumption that the circuits produce identical logits on the training dataset.

**Theorem D.4.** *Given  $I$  circuits  $C_i$  and associated  $L_q$  parameter norms  $P_i$ , assume that every circuit produces the same logits on the training dataset, i.e.  $\forall i, j, \forall (x, \_) \in \mathcal{D}, \forall y' \in Y$  we have  $\sigma_{C_i}^{y'}(x) = \sigma_{C_j}^{y'}(x)$ . Then, any weight vector  $\vec{w}^* \in \mathbb{R}^I$  that minimizes the loss in Equation 6 subject to  $w_i \geq 0$  satisfies:*

1. *If  $\kappa \geq q$ , then  $w_i^* = 0$  for all  $i$  such that  $P_i > \min_j P_j$ .*
2. *If  $0 < \kappa < q$ , then  $w_i^* \propto P_i^{-\frac{q\kappa}{q-\kappa}}$ .*

*Intuition.* Since every circuit produces identical logits, their weights are interchangeable with each other from the perspective of  $\mathcal{L}_{\text{x-ent}}$ , and so we must analyse how interchanging weights affects  $\mathcal{L}_{\text{wd}}$ .  $\mathcal{L}_{\text{wd}}$  grows as  $O(w_i^{2/\kappa})$ . When  $\kappa > 2$ ,  $\mathcal{L}_{\text{wd}}$  grows sublinearly, and so it is cheaper to add additional weight to the *largest* weight, creating a “rich get richer” effect that results in a single maximally efficient circuit getting all of the weight. When  $\kappa < 2$ ,  $\mathcal{L}_{\text{wd}}$  grows superlinearly, and so it is cheaper to add additional weight to the *smallest* weight. As a result, every circuit is allocated at least some weight, though more efficient circuits are still allocated higher weight than less efficient circuits.

*Sketch .* The assumption that every circuit produces the same logits on the training dataset implies that  $\mathcal{L}_{\text{train}}$  is purely a function of  $\sum_{i=1}^I w_i$ . So, for  $\mathcal{L}_{\text{train}}$ , a small increase  $\delta w$  to  $w_i$  can be balanced by a corresponding decrease  $\delta w$  to some other weight  $w_j$ .

For  $\mathcal{L}_{\text{wd}}$ , an increase  $\delta w$  to  $w_i$  produces a change of approximately  $\frac{\delta \mathcal{L}_{\text{wd}}}{\delta w_i} \cdot \delta w = \frac{\alpha}{\kappa} (P_i(w_i)^r)^q \cdot \delta w$ , where  $r = \frac{1}{\kappa} - \frac{1}{q} = \frac{q-\kappa}{q\kappa}$ . So, an increase of  $\delta w$  to  $w_i$  can be balanced by a decrease of  $\left(\frac{P_i(w_i)^r}{P_j(w_j)^r}\right)^q \delta w$  to some other weight  $w_j$ . The two cases correspond to  $r \leq 0$  and  $r > 0$  respectively.

**Case 1:  $r \leq 0$ .** Consider  $i, j$  with  $P_j > P_i$ . The optimal weights must satisfy  $w_i^* \geq w_j^*$  (else you could swap  $w_i^*$  and  $w_j^*$  to decrease loss). But then  $w_j^*$  must be zero: if not, we could increase  $w_i^*$  by  $\delta w$  and decrease  $w_j^*$  by  $\delta w$ , which keeps  $\mathcal{L}_{\text{x-ent}}$  constant and decreases  $\mathcal{L}_{\text{wd}}$  (since  $P_i(w_i^*)^r < P_j(w_j^*)^r$ ).

**Case 2:  $r > 0$ .** Consider  $i, j$  with  $P_j > P_i$ . As before we must have  $w_i^* \geq w_j^*$ . But now  $w_j^*$  must *not* be zero: otherwise we could increase  $w_j^*$  by  $\delta w$  and decrease  $w_i^*$  by  $\delta w$  to keep  $\mathcal{L}_{\text{x-ent}}$  constant and decrease  $\mathcal{L}_{\text{wd}}$ , since  $P_j(w_j^*)^r = 0 < P_i(w_i^*)^r$ . The balance occurs when  $P_j(w_j^*)^r = P_i(w_i^*)^r$ , which means  $w_i^* \propto P_i^{-1/r}$ .

*Proof.* First, notice that our conclusions trivially hold for  $\vec{w}^* = \vec{0}$  (which can be a minimum if e.g. the circuits are worse than random). Thus for the rest of the proof we will assume that at least one weight is non-zero.

In addition,  $\mathcal{L} \rightarrow \infty$  whenever any  $w_i \rightarrow \infty$  (because  $\mathcal{L}_{\text{train}} \geq B$  and  $\mathcal{L}_{\text{wd}} \rightarrow \infty$  as any one  $w_i \rightarrow \infty$ ). Thus, any global minimum must have finite  $\vec{w}$ .

Notice that, since the circuit logits are independent of  $i$ , we have  $h = (\sum_i w_i) f$ , and so  $\mathcal{L}_{\text{train}}(\vec{w})$  is purely a function of the sum of weights  $\sum_{i=1}^I w_i$ , and the overall loss can be written as:

$$\mathcal{L}(\vec{w}) = \mathcal{L}_{\text{train}}\left(\sum_{i=1}^I w_i\right) + \frac{\alpha}{q} \sum_{i=1}^I ((w_i)^{\frac{1}{\kappa}} P_i)^q$$

We will now consider each case in order.

**Case 1:  $\kappa \geq q$ .** Assume towards contradiction that there is a global minimum  $\vec{w}^*$  where  $w_j^* > 0$  for some circuit  $C_j$  with non-minimal  $P_j$ . Let  $C_i$  be a circuit with minimal  $P_i$  (so that  $P_i < P_j$ ), and let its weight be  $w_i^*$ .

Consider an alternate weight assignment  $\vec{w}'$  that is identical to  $\vec{w}^*$  except that  $w'_j = 0$  and  $w'_i = w_i^* + w_j^*$ . Clearly  $\sum_i w_i^* = \sum_i w'_i$ , and so  $\mathcal{L}_{\text{train}}(\vec{w}^*) = \mathcal{L}_{\text{train}}(\vec{w}')$ . Thus, we have:

$$\begin{aligned} & \mathcal{L}(\vec{w}^*) - \mathcal{L}(\vec{w}') \\ &= \left( \frac{\alpha}{q} \sum_{m=1}^I ((w_m^*)^{\frac{1}{\kappa}} P_m)^q \right) - \left( \frac{\alpha}{q} \sum_{m=1}^I ((w'_m)^{\frac{1}{\kappa}} P_m)^q \right) \\ &= \frac{\alpha}{q} \left( (w_i^*)^{\frac{q}{\kappa}} P_i^q + (w_j^*)^{\frac{q}{\kappa}} P_j^q - (w'_i)^{\frac{q}{\kappa}} P_i^q \right) \\ &> \frac{\alpha}{q} P_i^q \left( (w_i^*)^{\frac{q}{\kappa}} + (w_j^*)^{\frac{q}{\kappa}} - (w'_i)^{\frac{q}{\kappa}} \right) && \text{since } P_j > P_i \\ &= \frac{\alpha}{q} P_i^q \left( (w_i^*)^{\frac{q}{\kappa}} + (w_j^*)^{\frac{q}{\kappa}} - (w_i^* + w_j^*)^{\frac{q}{\kappa}} \right) && \text{definition of } w'_i \\ &\geq \frac{\alpha}{q} P_i^q \left( (w_i^*)^{\frac{q}{\kappa}} + (w_j^*)^{\frac{q}{\kappa}} - \left( (w_i^*)^{\frac{q}{\kappa}} + (w_j^*)^{\frac{q}{\kappa}} \right) \right) && \text{using Lemma D.2 since } 0 < \frac{q}{\kappa} \leq 1 \\ &= 0 \end{aligned}$$

Thus we have  $\mathcal{L}(\vec{w}^*) > \mathcal{L}(\vec{w}')$ , contradicting our assumption that  $\vec{w}^*$  is a global minimum of  $\mathcal{L}$ . This completes the proof for the case that  $\kappa \geq q$ .

**Case 2:**  $\kappa < q$ . First, we will show that all weights are non-zero at a global minimum (excluding the case where  $\vec{w}^* = \vec{0}$ , discussed at the beginning of the proof). Assume towards contradiction that there is a global minimum  $\vec{w}^*$  with  $w_j^* = 0$  for some  $j$ . Choose some arbitrary circuit  $C_i$  with nonzero weight  $w_i^*$ .

Choose some  $\epsilon_1 > 0$  satisfying  $\epsilon_1 < \frac{q}{2\kappa}(w_i^*)^{\frac{q}{\kappa}-1}$ . By applying Lemma D.3 with  $x = w_i^*$ ,  $c = \epsilon_1$ ,  $r = \frac{q}{\kappa}$ , we can get some  $\delta > 0$  such that for any  $\epsilon < \delta$  we have  $(w_i^*)^{\frac{q}{\kappa}} - (w_i^* - \epsilon)^{\frac{q}{\kappa}} > \delta(\frac{q}{\kappa}(w_i^*)^{\frac{q}{\kappa}-1} - \epsilon_1)$ .

Choose some  $\epsilon_2 > 0$  satisfying  $\epsilon_2 < \min(w_i^*, \delta, \left[\frac{q}{2\kappa}(w_i^*)^{\frac{q}{\kappa}-1} \frac{P_i^q}{P_j^q}\right]^{\frac{1}{\frac{q}{\kappa}-1}})$ . Consider an alternate weight assignment defined  $\vec{w}'$  that is identical to  $\vec{w}^*$  except that  $w_j' = \epsilon_2$  and  $w_i' = w_i^* - \epsilon_2$ . As in the previous case,  $\mathcal{L}_{\text{train}}(\vec{w}^*) = \mathcal{L}_{\text{train}}(\vec{w}')$ . Thus, we have:

$$\begin{aligned}
 & \mathcal{L}(\vec{w}^*) - \mathcal{L}(\vec{w}') \\
 &= \frac{\alpha}{q} \left( (w_i^*)^{\frac{q}{\kappa}} P_i^q - (w_i^* - \epsilon_2)^{\frac{q}{\kappa}} P_i^q - \epsilon_2^{\frac{q}{\kappa}} P_j^q \right) \\
 &= \frac{\alpha}{q} \left( P_i^q \left( (w_i^*)^{\frac{q}{\kappa}} - (w_i^* - \epsilon_2)^{\frac{q}{\kappa}} \right) - \epsilon_2^{\frac{q}{\kappa}} P_j^q \right) \\
 &> \frac{\alpha}{q} \left( P_i^q \delta \left( \frac{q}{\kappa}(w_i^*)^{\frac{q}{\kappa}-1} - \epsilon_1 \right) - \epsilon_2^{\frac{q}{\kappa}} P_j^q \right) && \text{application of Lemma D.3 discussed above} \\
 &> \frac{\alpha}{q} \left( P_i^q \delta \left( \frac{q}{\kappa}(w_i^*)^{\frac{q}{\kappa}-1} - \frac{q}{2\kappa}(w_i^*)^{\frac{q}{\kappa}-1} \right) - \epsilon_2^{\frac{q}{\kappa}} P_j^q \right) && \text{we chose } \epsilon_1 < \frac{q}{2\kappa}(w_i^*)^{\frac{q}{\kappa}-1} \\
 &> \frac{\alpha}{q} \left( P_i^q \epsilon_2 \frac{q}{2\kappa}(w_i^*)^{\frac{q}{\kappa}-1} - \epsilon_2^{\frac{q}{\kappa}} P_j^q \right) && \text{we chose } \epsilon_2 < \delta \\
 &= \frac{\alpha \epsilon_2}{q} \left( \frac{q}{2\kappa}(w_i^*)^{\frac{q}{\kappa}-1} P_i^q - \epsilon_2^{\frac{q}{\kappa}-1} P_j^q \right) \\
 &> \frac{\alpha \epsilon_2}{q} \left( \frac{q}{2\kappa}(w_i^*)^{\frac{q}{\kappa}-1} P_i^q - \frac{q}{2\kappa}(w_i^*)^{\frac{q}{\kappa}-1} \frac{P_i^q}{P_j^q} P_j^q \right) && \text{we chose } \epsilon_2 < \left[ \frac{q}{2\kappa}(w_i^*)^{\frac{q}{\kappa}-1} \frac{P_i^q}{P_j^q} \right]^{\frac{1}{\frac{q}{\kappa}-1}} \\
 &= 0
 \end{aligned}$$

Note that in the last step, we rely on the fact that  $\kappa < q$ : this lets us use an upper bound on  $\epsilon_2$  to get an upper bound on  $\epsilon_2^{\frac{q}{\kappa}-1}$ , and so a lower bound on the overall expression.

Thus we have  $\mathcal{L}(\vec{w}^*) > \mathcal{L}(\vec{w}')$ , contradicting our assumption that  $\vec{w}^*$  is a global minimum of  $\mathcal{L}$ . So, for all  $i$  we have  $w_i > 0$ .

In addition, as  $w_i \rightarrow \infty$  we have  $\mathcal{L}(\vec{w}) \rightarrow \infty$ , so  $\vec{w}^*$  cannot be at the boundaries, and instead lies in the interior. Since  $q > \kappa$ ,  $\mathcal{L}(\vec{w})$  is differentiable everywhere. Thus, we can conclude that its gradient

at  $\vec{w}^*$  is zero:

$$\begin{aligned}\frac{\delta \mathcal{L}}{\delta w_i} &= 0 \\ \frac{\delta \mathcal{L}_{\text{train}}}{\delta w_i} + \frac{\alpha P_i^q}{\kappa} (w_i^*)^{\frac{q}{\kappa}-1} &= 0 \\ P_i^q (w_i^*)^{\frac{q-\kappa}{\kappa}} &= -\frac{\kappa}{\alpha} \frac{\delta \mathcal{L}_{\text{train}}}{\delta w_i} \\ w_i^* P_i^{\frac{q\kappa}{q-\kappa}} &= \left( -\frac{\kappa}{\alpha} \frac{\delta \mathcal{L}_{\text{train}}}{\delta w_i} \right)^{\frac{\kappa}{q-\kappa}}\end{aligned}$$

Since  $\mathcal{L}_{\text{train}}(\vec{w})$  is a function of  $\sum_{j=1}^I w_j$ , we can conclude that  $\frac{\delta \mathcal{L}_{\text{train}}}{\delta w_i} = \frac{\delta \mathcal{L}_{\text{train}}}{\delta \sum_j w_j} \cdot \frac{\delta \sum_j w_j}{\delta w_i} = \frac{\delta \mathcal{L}_{\text{train}}}{\delta \sum_j w_j}$ , which is independent of  $i$ . So the right hand side of the equation is independent of  $i$ , allowing us to conclude that  $w_i^* \propto P_i^{-\frac{q\kappa}{q-\kappa}}$ .

□