On the training dynamics of deep networks with L_2 regularization

Aitor Lewkowycz
Google
Mountain View, CA
alewkowycz@google.com

Guy Gur-Ari
Google
Mountain View, CA
guyga@google.com

Abstract

We study the role of L_2 regularization in deep learning, and uncover simple relations between the performance of the model, the L_2 coefficient, the learning rate, and the number of training steps. These empirical relations hold when the network is overparameterized. They can be used to predict the optimal regularization parameter of a given model. In addition, based on these observations we propose a dynamical schedule for the regularization parameter that improves performance and speeds up training. We test these proposals in modern image classification settings. Finally, we show that these empirical relations can be understood theoretically in the context of infinitely wide networks. We derive the gradient flow dynamics of such networks, and compare the role of L_2 regularization in this context with that of linear models.

1 Introduction

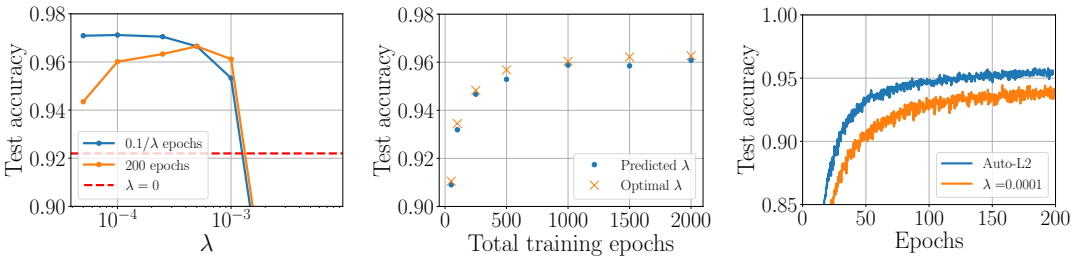
Machine learning models are commonly trained with L_2 regularization. This involves adding the term $\frac{1}{2}\lambda\|\theta\|_2^2$ to the loss function, where θ is the vector of model parameters and λ is a hyperparameter. In some cases, the theoretical motivation for using this type of regularization is clear. For example, in the context of linear regression, L_2 regularization increases the bias of the learned parameters while reducing their variance across instantiations of the training data; in other words, it is a manifestation of the bias-variance tradeoff. In statistical learning theory, a "hard" variant of L_2 regularization, in which one imposes the constraint $\|\theta\|_2 \leq \epsilon$, is often employed when deriving generalization bounds.

In deep learning, the use of L_2 regularization is prevalent and often leads to improved performance in practical settings [Hinton, 1986], although the theoretical motivation for its use is less clear. Indeed, it well known that overparameterized models overfit far less than one may expect [Zhang et al., 2016], and so the classical bias-variance tradeoff picture does not apply [Neyshabur et al., 2017, Belkin et al., 2018, Geiger et al., 2020]. There is growing understanding that this is caused, at least in part, by the (implicit) regularization properties of stochastic gradient descent (SGD) [Soudry et al., 2017]. The goal of this paper is to improve our understanding of the role of L_2 regularization in deep learning.

1.1 Our contribution

We study the role of L_2 regularization when training over-parameterized deep networks, taken here to mean networks that can achieve training accuracy 1 when trained with SGD. Specifically, we consider the early stopping performance of a model, namely the maximum test accuracy a model achieves during training, as a function of the L_2 parameter λ. We make the following observations based on the experimental results presented in the paper.

Preprint. Under review.



(a) λ independence (b) Optimal λ prediction (c) AUTOL_2 schedule

Figure 1: Wide ResNet 28-10 trained on CIFAR-10 with momentum and data augmentation. (a) Final test accuracy vs. the L_2 parameter λ. When the network is trained for a fixed amount of epochs, optimal performance is achieved at a certain value of λ. But when trained for a time proportional to λ^{-1}, performance plateaus and remains constant down to the lowest values of λ tested. This experiment includes a learning rate schedule. (b) Test accuracy vs. training epochs for predicted optimal L_2 parameter compared with the tuned parameter. (c) Training curves with our dynamical L_2 schedule, compared with a tuned, constant L_2 parameter.

1. The number of SGD steps until a model achieves maximum performance is $t_* \approx \frac{c}{\lambda}$, where c is a coefficient that depends on the data, the architecture, and all other hyperparameters. We find that this relationship holds across a wide range of λ values.
2. If we train with a fixed number of steps, model performance peaks at a certain value of the L_2 parameter. However, if we train for a number of steps proportional to λ^{-1} then performance improves with decreasing λ. In such a setup, performance becomes independent of λ for sufficiently small λ. Furthermore, performance with a small, non-zero λ is often better than performance without any L_2 regularization.

Figure 1a shows the performance of an overparameterized network as a function of the L_2 parameter λ. When the model is trained with a fixed steps budget, performance is maximized at one value of λ. However, when the training time is proportional to λ^{-1}, performance improves and approaches a constant value as we decrease λ.

As we demonstrate in the experimental section, these observations hold for a variety of training setups which include different architectures, data sets, and optimization algorithms. In particular, when training with vanilla SGD (without momentum), we observe that the number of steps until maximum performance depends on the learning rate η and on λ as $t_* \approx \frac{c'}{\eta \cdot \lambda}$. The performance achieved after this many steps depends only weakly on the choice of learning rate.

Applications. We present two practical applications of these observations. First, we propose a simple way to predict the optimal value of the L_2 parameter, based on a cheap measurement of the coefficient c. Figure 1b compares the performance of models trained with our predicted L_2 parameter with that of models trained with a tuned parameter. In this realistic setting, we find that our predicted parameter leads to performance that is within 0.4% of the tuned performance on CIFAR-10, at a cost that is marginally higher than a single training run. As shown below, we also find that the predicted parameter is consistently within an order of magnitude of the optimal, tuned value.

As a second application we propose AUTOL_2, a dynamical schedule for the L_2 parameter. The idea is that large L_2 values achieve worse performance but also lead to faster training. Therefore, in order to speed up training one can start with a large L_2 value and decay it during training (this is similar to the intuition behind learning rate schedules). In Figure 1c we compare the performance of a model trained with AUTOL_2 against that of a tuned but constant L_2 parameter, and find that AUTOL_2 outperforms the tuned model both in speed and in performance.

Theoretical contribution. Finally, we turn to a theoretical investigation of the empirical observations made above. As a first attempt at explaining these effects, consider the following argument based on the loss landscape. For overparameterized networks, the Hessian spectrum evolves rapidly during training [Sagun et al., 2017, Gur-Ari et al., 2018, Ghorbani et al., 2019]. After a small number of

training steps with no L_2 regularization, the minimum eigenvalue is found to be close to zero. In the presence of a small L_2 term, we therefore expect that the minimal eigenvalue will be approximately λ. In quadratic optimization, the convergence time is inversely proportional to the smallest eigenvalue of the Hessian.[1] Based on this intuition, we may then expect that convergence time will be proportional to λ^{-1}. The fact that performance is roughly constant for sufficiently small λ can then be explained if overfitting can be mostly attributed to optimization in the very low curvature directions [Rahaman et al., 2018, Wadia et al., 2020]. Now, our empirical finding is that the time it takes the network to reach maximum accuracy is proportional to λ^{-1}. In some cases this is the same as the convergence time, but in other cases (see for example Figure 4a) we find that performance decays after peaking and so convergence happens later. Therefore, the loss landscape-based explanation above is not sufficient to fully explain the effect.

To gain a better theoretical understanding, we consider the setup of an infinitely wide neural network trained using gradient flow. We focus on networks with positive-homogeneous activations, which include deep networks with ReLU activations, fully-connected or convolutional layers, and other common components. By analyzing the gradient flow update equations of such networks, we are able to show that the performance peaks at a time of order λ^{-1} and deteriorates thereafter. This is in contrast to the performance of linear models with L_2 regularization, where no such peak is evident. These results are consistent with our empirical observations, and may help shed light on the underlying causes of these effects.

According to known infinite width theory, in the absence of explicit regularization, the kernel that controls network training is constant [Jacot et al., 2018]. Our analysis extends the known results on infinitely wide network optimization, and indicates that the kernel decays in a predictable way in the presence of L_2 regularization. We hope that this analysis will shed further light on the observed performance gap between infinitely wide networks which are under good theoretical control, and the networks trained in practical settings [Arora et al., 2019, Novak et al., 2019, Wei et al., 2018, Lewkowycz et al., 2020].

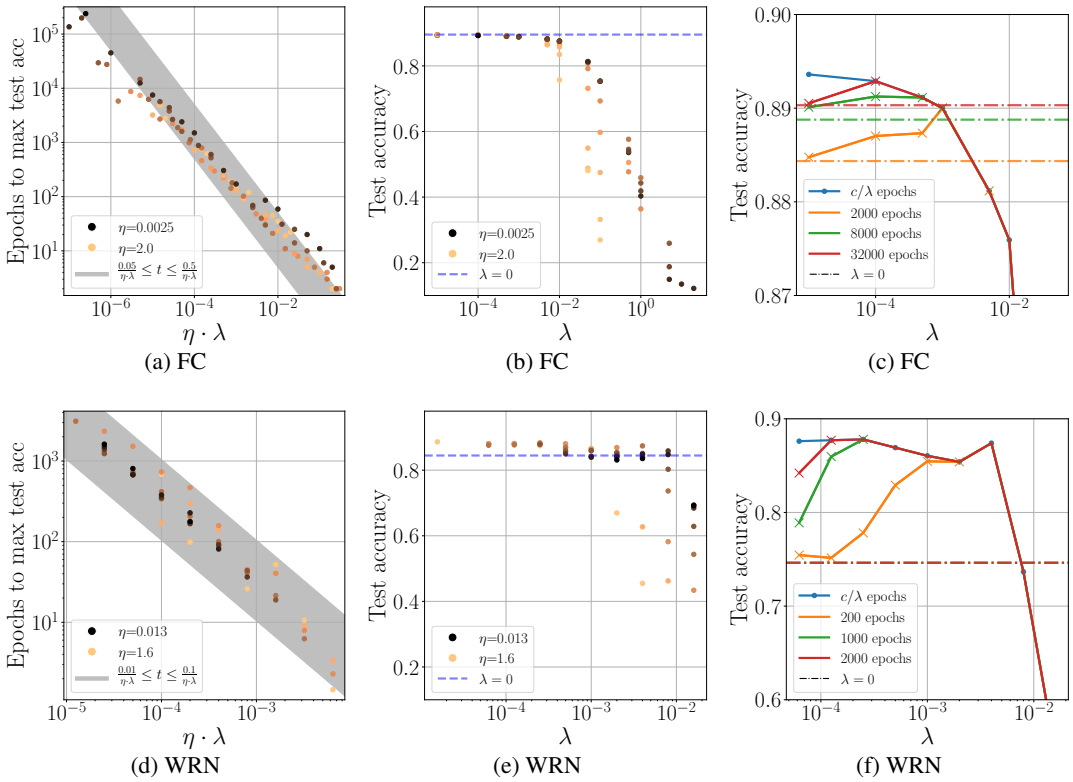
Related works. L_2 regularization in the presence of batch-normalization [Ioffe and Szegedy, 2015] has been studied in [van Laarhoven, 2017, Hoffer et al., 2018, Zhang et al., 2018]. These papers discussed how the effect of L_2 on scale invariant models is merely of having an effective learning rate (and no L_2). This was made precise in Li and Arora [2019] where they showed that this effective learning rate is $\eta_{\text{eff}} = \eta e^{2\eta\lambda t}$ (at small learning rates). Our theoretical analysis of large width networks will have has the same behaviour when the network is scale invariant.

2 Experiments

Performance and time scales. We now turn to an empirical study of networks trained with L_2 regularization. In this section we present results for a fully-connected network trained on MNIST, a Wide ResNet [Zagoruyko and Komodakis, 2016] trained on CIFAR-10, and CNNs trained on CIFAR-10. The experimental details are in SM A. The empirical findings discussed in section 1.1 hold across this variety of overparameterized setups.

Figure 2 presents experimental results on fully-connected and Wide ResNet networks. Figure 3 presents experiments conducted on CNNs. We find that the number of steps until optimal performance is achieved (defined here as the minimum time required to be within $.5\%$ of the maximum test accuracy) scales as λ^{-1}, as discussed in Section 1.1. Our experiments span 6 decades of $\eta \cdot \lambda$ (larger η, λ won't train at all and smaller would take too long to train). Moreover, when we evolved the networks until they have reached optimal performance, the maximum test accuracy for smaller L_2 parameters did not get worse. We compare this against the performance of a model trained with a fixed number of epochs, reporting the maximum performance achieved during training. In this case, we find that reducing λ beyond a certain value does hurt performance.

[1] In linear regression with L_2 regularization, optimization is controlled by a linear kernel $K = X^T X + \lambda I$, where X is the sample matrix and I is the identity matrix in parameter space. Optimization in each kernel eigendirection evolves as $e^{-\gamma t}$ where γ is the corresponding eigenvalue. When $\lambda > 0$ and the model is overparameterized, the lowest eigenvalue of the kernel will be typically close to λ, and therefore the time to convergence will be proportional to λ^{-1}.



(a) FC (b) FC (c) FC

(d) WRN (e) WRN (f) WRN

Figure 2: Sweep over η and λ illustrating how smaller λ's require longer times to achieve the same performance. In the left, middle plots, the learning rates are logarithmically spaced between the values displayed in the legend, the specific values are in the SM A. **Left:** Epochs to maximum test accuracy (within .5%), **Middle:** Maximum test accuracy (the $\lambda = 0$ line denotes the maximum test accuracy achieved among all learning rates), **Right:** Maximum test accuracy for a fixed time budget. **(a,b,c)** Fully connected 3-hidden layer neural network evaluated in 512 MNIST samples, evolved for $t \cdot \eta \cdot \lambda = 2$. $\eta = 0.15$ in (c). **(d,e,f)** A Wide Residual Network 28-10 trained on CIFAR-10 without data augmentation, evolved for $t \cdot \eta \cdot \lambda = 0.1$. In (f), $\eta = 0.2$. The $\lambda = 0$ line was evolved for longer than the smallest L_2 but there is still a gap.

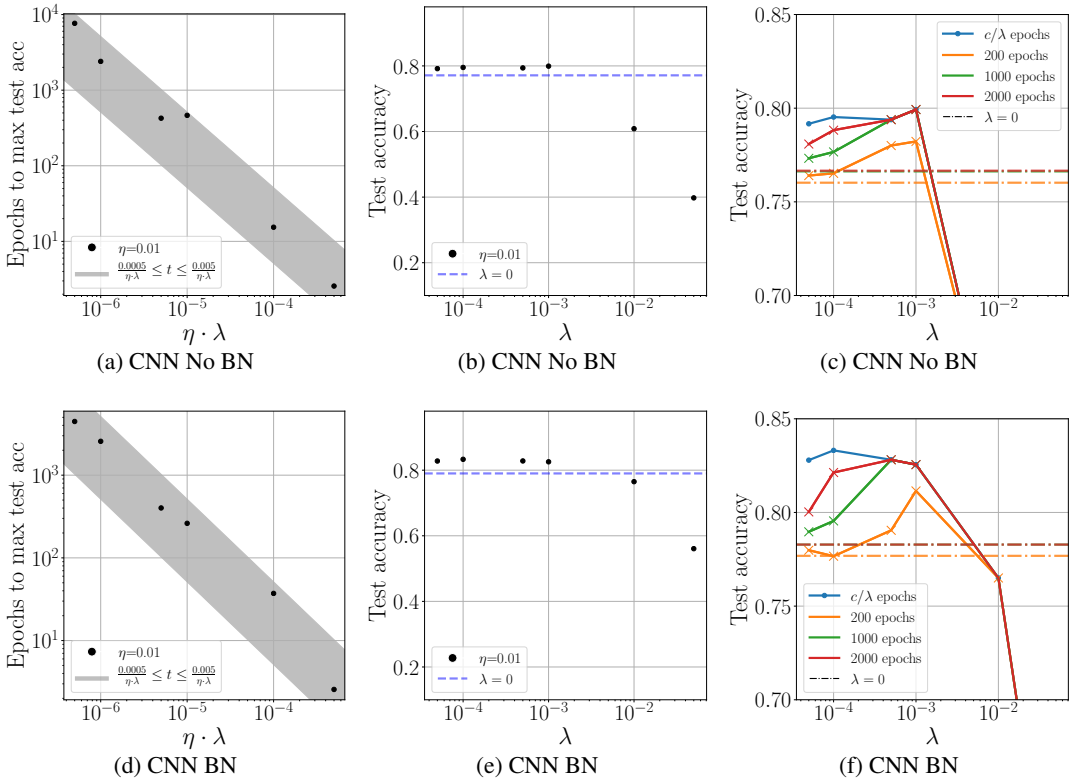
While here we consider the simplified set up of vanilla SGD and no data augmentation, our observations also hold in the presence of momentum and data augmentation, see SM C.2 for more experiments. We would like to emphasize again that while the smaller L_2 models can reach the same test accuracy as its larger counterparts, models like WRN28-10 on CIFAR-10 need to be trained for a considerably larger number of epochs to achieve this.[2]

Learning rate schedules. So far we considered training setups that do not include learning rate schedules. Figure 1a shows the results of training a Wide ResNet on CIFAR-10 with a learning rate schedule, momentum, and data augmentation. The schedule was determined as follows. Given a total number of epochs T, the learning rate is decayed by a factor of 0.2 at epochs $\{0.3 \cdot T, 0.6 \cdot T, 0.9 \cdot T\}$. We compare training with a fixed T against training with $T \propto \lambda^{-1}$. We find that training with a fixed budget leads to an optimal value of λ, below which performance degrades. On the other hand, training with $T \propto \lambda^{-1}$ leads to improved performance at smaller λ, consistent with our previous observations.

3 Applications

We now discuss two practical applications of the empirical observations made in the previous section.

[2]The longer experiments ran for 5000 epochs while one usually trains these models for ∼300 epochs.



(a) CNN No BN (b) CNN No BN (c) CNN No BN

(d) CNN BN (e) CNN BN (f) CNN BN

Figure 3: CNNs trained with and without batch-norm with learning rate $\eta = 0.01$. Presented results follow the same format as Figure 2.

Optimal L_2. We observed that the time t_* to reach maximum test accuracy is proportional to λ^{-1}, which we can express as $t_* \approx \frac{c}{\lambda}$. This relationship continues to hold empirically even for large values of λ. When λ is large, the network attains its (significantly degraded) maximum performance after a relatively short amount of training time. We can therefore measure the value of c by training the network with a large L_2 parameter until its performance peaks, at a fraction of the cost of a normal training run.

Based on our empirical observations, given a training budget T we predict that the optimal L_2 parameter can be approximated by $\lambda_{\text{pred}} = c/T$. This is the smallest L_2 parameter such that model performance will peak within training time T. Figure 1b shows the result of testing this prediction in a realistic setting: a Wide ResNet trained on CIFAR-10 with momentum$= 0.9$, learning rate $\eta = 0.2$ and data augmentation. The model is first trained with a large L_2 parameter for 2 epochs in order to measure c, and we find $c \approx 0.0066$, see figure 4a. We then compare the tuned value of λ against our prediction for training budgets spanning close to two orders of magnitude, and find excellent agreement: the predicted λ's have a performance which is rather close to the optimal one. Furthermore, the tuned values are always within an order of magnitude of our predictions see figure 4b.

We have not studied this thoroughly in the presence of learning rate schedules, but intuitively, one wants to evolve for as long as possible with the initial large learning rate and it seems natural to think that the optimal λ corresponds to the smallest L_2 which makes it to the (constant learning rate) peak before the first decay. Using the learning rate schedule of section 2, this implies $\lambda_{\text{pred}} \approx \frac{c}{0.3T}$. In the setup of $T = 200$ epochs of figure 1a, this predicts $\lambda_{\text{pred}} \approx 0.0001$ (test accuracy 0.960), which is close to the optimal at 0.0005 (with test accuracy 0.967). While the predicted L_2 is close, this has substantial implications for performance, but we can nevertheless use it as a reference for hyperparameter tuning. We leave a more precise estimation of the optimal L_2 for future work.

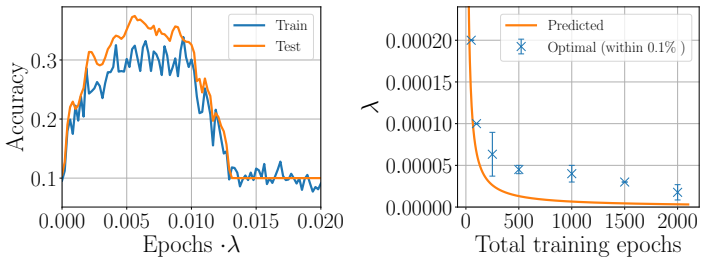


Figure 4: Wide ResNet trained with momentum and data augmentation. (a) We train the model with a large L_2 parameter $\lambda = 0.01$ for 2 epochs and measure the coefficient $c = t_* \cdot \lambda \approx 0.0066$, representing the approximate point along the x axis where accuracy is maximized. (b) Optimal (tuned) λ values compared with the theoretical prediction. The error bars represent the spread of values that achieve within 0.1% of the optimal test accuracy.

AUTOL_2: Automatic L_2 schedules. We now turn to another application, based on the observation that models trained with larger L_2 parameters reach their peak performance faster. It is therefore plausible that one can speed up the training process by starting with a large L_2 parameter, and decaying it according to some schedule. Here we propose to choose the schedule dynamically by decaying the L_2 parameter when performance begins to deteriorate. See SM E for further details.

AUTOL_2 is a straightforward implementation of this idea: We begin training with a large parameter, $\lambda = 0.1$, and we decay it by a factor of 10 if either the empirical loss (the training loss without the L_2 term) or the training error increases. To improve stability, immediately after decaying we impose a refractory period during which the parameter cannot decay again. Figure 1c compares this algorithm against the model with the optimal L_2 parameter. We find that AUTOL_2 trains significantly faster and achieves superior performance.

In other experiments we have found that this algorithm does not yield improved results when the training procedure includes a learning rate schedule. We leave the attempt to effectively combine learning rate schedules with L_2 schedules to future work.

4 Theoretical results

We now turn to a theoretical analysis of the training trajectory of networks trained with L_2 regularization. We focus on infinitely wide networks with positively-homogeneous activations. Consider a network function $f : \mathbb{R}^d \to \mathbb{R}$ with model parameter $\theta \in \mathbb{R}^p$. The network initialized using NTK parameterization [Jacot et al., 2018]: the initial parameters are sampled i.i.d. from $\mathcal{N}(0, 1)$. The model parameters are trained using gradient flow with loss $L_{\text{tot}} = L + \frac{\lambda}{2}\|\theta\|_2^2$, where $L = \sum_{(x,y)\in S} \ell(x, y)$ is the empirical loss, ℓ is the sample loss, and S is the training set of size N_{samp}.

We say that the network function is k-homogeneous if $f_{\alpha\theta}(x) = \alpha^k f_\theta(x)$ for any $\alpha > 0$. As an example, a fully-connected network with L layers and ReLU or linear activations is L-homogeneous. Networks made out of convolutional, max-pooling or batch-normalization layers are also k-homogeneous.[3] See Li and Arora [2019] for a discussion of networks with homogeneous activations.

Jacot et al. [2018] showed that when an infinitely wide, fully-connected network is trained using gradient flow (and without L_2 regularization), its network function obeys the differential equation $\frac{df}{dt}(x) = -\sum_{x'\in S} \Theta_0(x, x')\ell'(x')$, where t is the gradient flow time and $\Theta_t(x, x') = \nabla_\theta f_t(x)^T \nabla_\theta f_t(x)$ is the Neural Tangent Kernel (NTK).

Dyer and Gur-Ari [2020] presented a conjecture that allows one to derive the large width asymptotic behavior of the network function, the Neural Tangent Kernel, as well as of combinations involving

[3] Batch normalization is often implemented with an ϵ parameter meant to prevent numerical instabilities. Such networks are only approximately homogeneous.

higher-order derivatives of the network function. The conjecture was shown to hold for networks with polynomial activations [Aitken and Gur-Ari, 2020], and has been verified empirically for commonly used activation functions. In what follows, we will assume the validity of this conjecture. The following is our main theoretical result.

Theorem 1. *Consider a k-homogeneous network, and assume that the network obeys the correlation function conjecture of Dyer and Gur-Ari [2020]. In the infinite width limit, the network function $f_t(x)$ and the kernel $\Theta_t(x, x')$ evolve according to the following equations at training time t.*

$$\frac{df_t(x)}{dt} = -e^{-2(k-1)\lambda t} \sum_{(x',y')\in S} \Theta_0(x, x') \frac{\partial \ell(x', y')}{\partial f_t} - \lambda k f_t(x)\,, \tag{1}$$

$$\frac{d\Theta_t(x, x')}{dt} = -2(k-1)\lambda \Theta_t(x, x')\,. \tag{2}$$

The proof hinges on the following equation, which holds for k-homogeneous functions: $\sum_\mu \theta_\mu \partial_\mu \partial_{\nu_1} \cdots \partial_{\nu_m} f(x) = (k-m)\partial_{\nu_1} \cdots \partial_{\nu_m} f(x)$. This equation allows us to show that the only effect of L_2 regularization at infinite width is to introduce simple terms proportional to λ in the gradient flow update equations for both the function and the kernel.

We refer the reader to the SM for the proof. We mention in passing that the case $k = 0$ corresponds to a scaling-invariant network function which was studied in Li and Arora [2019]. In this case, training with L_2 term is equivalent to training with an exponentially increasing learning rate.

For commonly used loss functions, and for $k > 1$, we expect that the solution obeys $\lim_{t\to\infty} f_t(x) = 0$. We will prove that this holds for MSE loss, but let us first discuss the intuition behind this statement. At late times the exponent in front of the first term in (1) decays to zero, leaving the approximate equation $\frac{df(x)}{dt} \approx -\lambda k f(x)$ and leading to an exponential decay of the function to zero. Both the explicit exponent in the equation, and the approximate late time exponential decay, suggest that this decay occurs at a time $t_{\rm decay} \propto \lambda^{-1}$. Therefore, we expect that the minimum of the empirical loss to occur at a time proportional to λ^{-1}, after which the bare loss will increase because the function is decaying to zero. We observe this behaviour empirically for wide fully-connected networks and for Wide ResNet in the SM.

Furthermore, notice that if we include the k dependence, the decay time scale is approximately $t_{\rm decay} \propto (k\lambda)^{-1}$. Models with a higher degree of homogeneity (for example deeper fully-connected networks) will converge faster.

We now focus on MSE loss and solve the gradient flow equation (1) for this case.

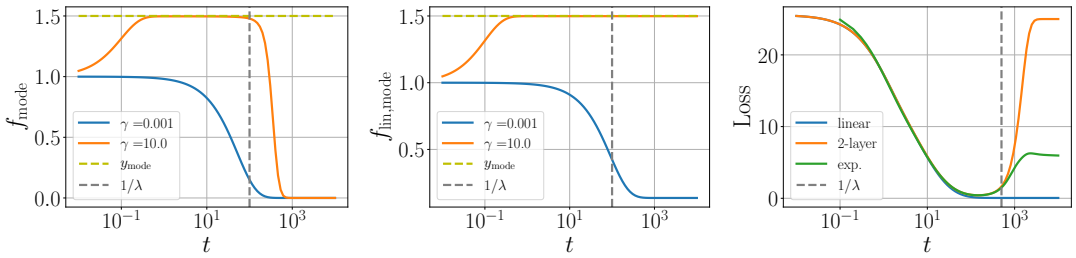
Theorem 2. *Let the sample loss be $\ell(x, y) = \frac{1}{2}(f(x) - y)^2$, and assume that $k \geq 2$. Suppose that, at initialization, the kernel Θ_0 has eigenvectors $\hat{e}_a \in \mathbb{R}^{N_{\rm samp}}$ with corresponding eigenvalues γ_a. Then during gradient flow, the eigenvalues evolve as $\gamma_a(t) = \gamma_a e^{-2(k-1)\lambda t}$ while the eigenvectors are static. Suppose we treat $f \in \mathbb{R}^{N_{\rm samp}}$ as a vector defined on the training set. Then each mode of the function, $f_a := (\hat{e}_a)^T f \in \mathbb{R}$, evolves independently as*

$$f_a(x; t) = e^{\frac{\gamma_a(t)}{2(k-1)\lambda} - k\lambda t} \left\{ e^{-\frac{\gamma_a}{2(k-1)\lambda}} f_a(x; 0) + \gamma_a y_a \int_0^t dt' \, \exp\left[-\frac{\gamma_a(t')}{2(k-1)\lambda} - (k-2)\lambda t' \right] \right\}. \tag{3}$$

Here, $y_a := (\hat{e}_a)^T y$. At late times, $\lim_{t\to\infty} f_t(x) = 0$ on the training set.

The properties of the solution (3) depend on whether the ratio γ_a/λ is greater than or smaller than 1, as illustrated in Figure 5. When $\gamma_a/\lambda > 1$, the function approaches the label mode $y_{\rm mode} = y_a$ at a time that is of order $1/\gamma_a$. This behavior is the same as that of a linear model, and represents ordinary learning. Later, at a time of order λ^{-1} the mode decays to zero as described above; this late time decay is not present in the linear model. Next, when $\gamma_a/\lambda < 1$ the mode decays to zero at a time of order λ^{-1}, which is the same behavior as that of a linear model.

Generalization of wide networks with L_2. It is interesting to understand how L_2 regularization affects the generalization performance of wide networks. This is well understood for the case of linear models, which correspond to $k = 1$ in our notation, to be an instance of the bias-variance



(a) 2-layer network (b) linear model (c) losses

Figure 5: (a) The theoretical evolution of an infinitely wide 2-layer network with L_2 regularization ($k = 2$, $\lambda = 0.01$). Two modes are shown, representing small and large ratios γ/λ. (b) The same, for a linear model ($k = 1$). (c) Training loss vs. time for a wide network trained on a subset of MNIST with even/odd labels, with $\lambda = 0.002$. We compare the kernel evolution with gradient descent for a 2-layer ReLU network. The blue and orange curves are the theoretical predictions when setting $k = 1$ and $k = 2$ in the solution (3), respectively. The green curve is the result of a numerical experiment where we train a 2-layer ReLU network with gradient descent. We attribute the difference between the green and orange curves at late times to finite width effects.

tradeoff. In this case, gradient flow converges to the function $f_*(x) = \Theta(x, X)(\Theta + \lambda I)^{-1}(X, X)Y$, where $X \in \mathbb{R}^{N_{\rm samp} \times d}$ are the training samples, $Y \in \mathbb{R}^{N_{\rm samp}}$ is are the labels, and $x \in \mathbb{R}^d$ is any input. When $\lambda = 0$, the solution is highly sensitive to small perturbations in the inputs that affect the flat modes of the kernel, because the kernel is inverted in the solution. In other words, the solution has high variance. Choosing $\lambda > 0$ reduces variance by lifting the low kernel eigenvalues and reducing sensitivity on small perturbations, at the cost of biasing the model parameters toward zero. While a linear model is the prototypical case of $k = 1$, the previous late time solution $f_*(x)$ is valid for any $k = 1$ model. In particular, any homogeneous model that has batch-normalization in the pre-logit layer will satisfy this property. It would be interesting to understand the generalization properties of such these models based on these solutions.

Let us now return to infinitely wide networks. These behave like linear models with a fixed kernel when $\lambda = 0$, but as we have seen when $\lambda > 0$ the kernel decays exponentially. Nevertheless, we argue that this decay is slow enough such that the training dynamics follow that of the linear model (obtained by setting $k = 1$ in eq. (1)) up until a time of order λ^{-1}, when the function begins decaying to zero. This can be seen in Figure 5c, which compares the training curves of a linear and a 2-layer network using the same kernel. We see that the agreement extends until the linear model is almost fully trained, at which point the 2-layer model begins deteriorating due to the late time decay. Therefore, if we stop training the 2-layer network at the loss minimum, we end up with a trained and regularized model. It would be interesting to understand how the generalization properties of this model with decaying kernel differ from those of the linear model.

Finite-width network. Theorem 1 holds in the strict large width, fixed λ limit for NTK parameterization. At large but finite width we expect (1) to be a good description of the training trajectory at early times, until the kernel and function because small enough such that the finite-width corrections become non-negligible. Our experimental results imply that this approximation remains good until after the minimum in the loss, but that at late times the function will not decay to zero; see for example Figure 5c. See the SM for further discussion for the case of deep linear models. We reserve a more careful study of these finite width effects to future work.

5 Discussion

In this work we consider the effect of L_2 regularization on overparameterized networks. We make two empirical observations: (1) The time it takes the network to reach peak performance is proportional to λ, the L_2 regularization parameter, and (2) the performance reached in this way is independent of λ when λ is not too large. We find that these observations hold for a variety of overparameterized training setups; see the SM for some examples where they do not hold.

Motivated by these observations, we suggest two practical applications. The first is a simple method for predicting the optimal L_2 parameter at a given training budget. The performance obtained using this prediction is close to that of a tuned L_2 parameter, at a fraction of the training cost. The second is AUTOL_2, an automatic L_2 parameter schedule. In our experiments, this method leads to better performance and faster training when compared against training with a tuned L_2 parameter. We find that these proposals work well when training with a constant learning rate; we leave an extension of these methods to networks trained with learning rate schedules to future work.

We attempt to understand the empirical observations by analyzing the training trajectory of infinitely wide networks trained with L_2 regularization. We derive the differential equations governing this trajectory, and solve them explicitly for MSE loss. The solution reproduces the observation that the time to peak performance is of order λ^{-1}. This is due to an effect that is specific to deep networks, and is not present in linear models: during training, the kernel (which is constant for linear models) decays exponentially due to the L_2 term.

Acknowledgments

The authors would like to thank Yasaman Bahri, Ethan Dyer, Jaehoon Lee and Behnam Neyshabur for useful discussions. We specially thank Behnam for encouraging us to use the scaling to come up with an L_2 schedule.

References

Kyle Aitken and Guy Gur-Ari. To appear. 2020.

Sanjeev Arora, Simon S Du, Wei Hu, Zhiyuan Li, Russ R Salakhutdinov, and Ruosong Wang. On exact computation with an infinitely wide neural net. In *Advances in Neural Information Processing Systems*, pages 8139–8148, 2019.

Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mand al. Reconciling modern machine learning practice and the bias-variance trade-off. *arXiv e-prints*, art. arXiv:1812.11118, December 2018.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, and Skye Wanderman-Milne. JAX: composable transformations of Python+NumPy programs. 2018. URL `http://github.com/google/jax`.

Ethan Dyer and Guy Gur-Ari. Asymptotics of wide networks from feynman diagrams. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=S1gFvANKDS`.

Mario Geiger, Arthur Jacot, Stefano Spigler, Franck Gabriel, Levent Sagun, Stéphane d'Ascoli, Giulio Biroli, Clément Hongler, and Matthieu Wyart. Scaling description of generalization with number of parameters in deep learning. *Journal of Statistical Mechanics: Theory and Experiment*, 2(2):023401, February 2020. doi: 10.1088/1742-5468/ab633c.

Behrooz Ghorbani, Shankar Krishnan, and Ying Xiao. An Investigation into Neural Net Optimization via Hessian Eigenvalue Density. *arXiv e-prints*, art. arXiv:1901.10159, January 2019.

Guy Gur-Ari, Daniel A. Roberts, and Ethan Dyer. Gradient Descent Happens in a Tiny Subspace. *arXiv e-prints*, art. arXiv:1812.04754, December 2018.

G. E. Hinton. Learning distributed representations of concepts. *Proc. of Eighth Annual Conference of the Cognitive Science Society, 1986*, 1986.

Elad Hoffer, Ron Banner, Itay Golan, and Daniel Soudry. Norm matters: efficient and accurate normalization schemes in deep networks, 2018.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015. URL `http://arxiv.org/abs/1502.03167`.

Arthur Jacot, Franck Gabriel, and Clement Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 8571–8580. Curran Associates, Inc., 2018.

Aitor Lewkowycz, Yasaman Bahri, Ethan Dyer, Jascha Sohl-Dickstein, and Guy Gur-Ari. The large learning rate phase of deep learning: the catapult mechanism, 2020.

Zhiyuan Li and Sanjeev Arora. An exponential learning rate schedule for deep learning, 2019.

Behnam Neyshabur, Srinadh Bhojanapalli, David McAllester, and Nathan Srebro. Exploring Generalization in Deep Learning. *arXiv e-prints*, art. arXiv:1706.08947, June 2017.

Roman Novak, Lechao Xiao, Yasaman Bahri, Jaehoon Lee, Greg Yang, Daniel A. Abolafia, Jeffrey Pennington, and Jascha Sohl-dickstein. Bayesian deep convolutional networks with many channels are gaussian processes. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=B1g30j0qF7`.

Nasim Rahaman, Aristide Baratin, Devansh Arpit, Felix Draxler, Min Lin, Fred A Hamprecht, Yoshua Bengio, and Aaron Courville. On the spectral bias of neural networks. *arXiv preprint arXiv:1806.08734*, 2018.

Levent Sagun, Utku Evci, V. Ugur Guney, Yann Dauphin, and Leon Bottou. Empirical Analysis of the Hessian of Over-Parametrized Neural Networks. *arXiv e-prints*, art. arXiv:1706.04454, June 2017.

Daniel Soudry, Elad Hoffer, Mor Shpigel Nacson, Suriya Gunasekar, and Nathan Srebro. The implicit bias of gradient descent on separable data, 2017.

Twan van Laarhoven. L2 regularization versus batch and weight normalization, 2017.

Neha Wadia, Daniel Duckworth, Sam Schoenholtz, Ethan Dyer, and Jascha Sohl-dickstein. To appear., 2020.

Colin Wei, Jason D. Lee, Qiang Liu, and Tengyu Ma. Regularization matters: Generalization and optimization of neural nets v.s. their induced kernel, 2018.

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *CoRR*, abs/1605.07146, 2016. URL `http://arxiv.org/abs/1605.07146`.

Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *arXiv e-prints*, art. arXiv:1611.03530, November 2016.

Guodong Zhang, Chaoqi Wang, Bowen Xu, and Roger Grosse. Three mechanisms of weight decay regularization, 2018.

Supplementary material

A Experimental details

We are using JAX [Bradbury et al., 2018].

All the models except for section C.4 have been trained with Softmax loss normalized as $\mathcal{L}(\{x,y\}_B) = \frac{1}{2k|B|}\sum_{(x,y)\in B,i} y_i \log p_i(x), p_i(x) = \frac{e^{f^i(x)}}{\sum_j e^{f^j(x)}}$, where k is the number of classes and y^i are one-hot targets.

All experiments that compare different learning rates and L_2 parameters use the same seed for the weights at initialization and we consider only one such initialization (unless otherwise stated) although we have not seen much variance in the phenomena described. We will be using standard normalization with LeCun initialization $W \sim \mathcal{N}(0, \frac{\sigma_w^2}{N_{in}}), b \sim \mathcal{N}(0, \sigma_b^2)$.

Batch Norm: we are using JAX's Stax implementation of Batch Norm which doesn't keep track of training batch statistics for test mode evaluation.

Data augmentation: denotes flip, crop and mixup.

We consider 3 different networks:

- WRN: Wide Resnet 28-10 [Zagoruyko and Komodakis, 2016] with has batch-normalization and batch size 1024 (per device batch size of 128), $\sigma_w = 1, \sigma_b = 0$. Trained on CIFAR-10.
- FC: Fully connected, three hidden layers with width 2048 and ReLU activation and batch size 512,$\sigma_w = \sqrt{2}, \sigma_b = 0$. Trained on 512 samples of MNIST.
- CNN: We use the following architecture: $\text{Conv}_1(300) \to$ Act $\to \text{Conv}_2(300) \to$ Act $\to$ MaxPool((6,6), 'VALID') $\to \text{Conv}_1(300) \to$ Act $\to \text{Conv}_2(300) \to$ MaxPool((6,6), 'VALID') $\to$ Flatten() $\to$ Dense$(500) \to$ Dense(10). Dense(n) denotes a fully-connected layer with output dimension n. $\text{Conv}_1(n), \text{Conv}_2(n)$ denote convolutional layers with 'SAME' or 'VALID' padding and n filters, respectively; all convolutional layers use $(3,3)$ filters. MaxPool((2,2), 'VALID') performs max pooling with 'VALID' padding and a (2,2) window size. Act denotes the activation: '(Batch-Norm $\to$) ReLU ' depending on whether we use Batch-Normalization or not. We use batch size 128, $\sigma_w = \sqrt{2}, \sigma_b = 0$. Trained on CIFAR-10 without data augmentation.

The WRN experiments are run on v3-8 TPUs and the rest on P100 GPUs.

Here we describe the particularities of each figure. Whenever we report performance for a given time budget, we report the maximum performance during training which does not have to happen at the end of training.

Figure 1a WRN trained using momentum$= 0.9$, data augmentation and a learning rate schedule where $\eta(t=0) = 0.2$ and then decays $\eta \to 0.2\eta$ at $\{0.3\cdot T, 0.6\cdot T, 0.9\cdot T\}$, where T is the number of epochs. We compare training with a fixed $T = 200$ training budget, against training with $T(\lambda) = 0.1/\lambda$. This was chosen so that $T(0.0005) = 200$.

Figures 1b, 4, S4. WRN trained using momentum$= 0.9$, data augmentation and $\eta = 0.2$ for $\lambda \in (5\cdot 10^{-6}, 10^{-5}, 5\cdot 10^{-5}, 0.0001, 0.0002, 0.0004, 0.001, 0.002)$. The predicted λ performance of 1b was computed at $\lambda = 0.0066/T \in (0.000131, 6.56\cdot 10^{-5}, 2.63\cdot 10^{-5}, 1.31\cdot 10^{-5}, 6.56\cdot 10^{-6}, 4.38\cdot 10^{-6}, 3.28\cdot 10^{-6})$ for $T \in (50, 100, 250, 500, 1000, 1500, 2000)$ respectively.

Figures 1c,S9. WRN trained using momentum$= 0.9$, data augmentation and $\eta = 0.2$, evolved for 200 epochs. The AUTOL_2 algorithm is written explicitly in SM E and make measurements every 10 steps.

Figure 2a,b,c. FC trained using SGD $\frac{2}{\eta\lambda}$ epochs with learning rate and L_2 regularizations $\eta \in (0.0025, 0.01, 0.02, 0.025, 0.03, 0.05, 0.08, 0.15, 0.3, 0.5, 1, 1.5, 2, 5, 10, 25, 50)$, $\lambda \in (0, 10^{-5}, 0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 5, 10, 20, 50, 100)$. The $\lambda = 0$ model was evolved for $10^6/\eta$ epochs which is more than the smallest λ.

Figure 2d,e,f. WRN trained using SGD without data augmentation for $\frac{0.1}{\eta\lambda}$ epochs for the following hyperparameters $\eta \in (0.0125, 0.025, 0.05, 0.1, 0.2, 0.4, 0.8, 1.6)$, $\lambda \in (0, 1.5625 \cdot 10^{-5}, 6.25 \cdot 10^{-5}, 1.25 \cdot 10^{-4}, 2.5 \cdot 10^{-4}, 5 \cdot 10^{-4}, 10^{-3}, 2 \cdot 10^{-3}, 4 \cdot 10^{-3}, 8 \cdot 10^{-3}, 0.016)$, as long as the total number of epochs was ≤ 4000 epochs (except for $\eta = 0.2, \lambda = 6.25 \cdot 10^{-5}$ which was evolved for 8000 epochs). We evolved the $\lambda = 0$ models for 10000 epochs.

Figure S7. Fully connected depth 3 and width 64 trained on CIFAR-10 with batch size 512, $\eta = 0.1$ and cross-entropy loss.

Figure S8. ResNet-50 trained on ImageNet with batch size 8192, using the implementation in `https://github.com/tensorflow/tpu`.

Figure 5 (a,b) plots f_t in equation 3 with $k = 2$ (for 2-layer) and $k = 1$ (for linear), for different values of γ and $\lambda = 0.01$. (c) The empirical kernel of a $2-$layer ReLU network of width 5,000 was evaluated on 200-samples of MNIST with even/odd labels. The linear, $2-$layer curves come from evolving equation 1 with the previous kernel and setting $k = 1, k = 2$, respectively . The experimental curve comes from training the 2-layer ReLU network with width 10^5 and learning rate $\eta = 0.01$ (the time is step $\times\, \eta$).

Figure 3a,b,c. CNN without BN trained using SGD for $\frac{0.01}{\eta\lambda}$ epochs for the following hyperparameters $\eta = 0.01, \lambda \in (0, 5 \cdot 10^{-5}, 0.0001, 0.0005, 0.001, 0.01, 0.05, 0.1, 0.25, 0.5, 1, 2)$. with $\lambda = 0$ was evolved for 21000 epochs.

Figure 3d,e,f. CNN with BN trained using SGD for a time $\frac{0.01}{\eta\lambda}$ for the following hyperparameters $\eta = 0.01, \lambda = 0, 5 \cdot 10^{-5}, 0.0001, 0.0005, 0.001, 0.01, 0.05, 0.1, 0.25, 0.5, 1, 2$. The model with $\lambda = 0$ was evolved for 9500 epochs, which goes beyond where all the other λ's have peaked.

Figure S5. FC trained using SGD and MSE loss for $\frac{1}{\eta\lambda}$ epochs and the following hyperparameters $\eta \in (0.001, 0.005, 0.01, 0.02, 0.035, 0.05, 0.15, 0.3), \lambda \in (10^{-5}, 0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 5, 50, 100)$. For $\lambda = 0$, it was trained for $10^5/\eta$ epochs.

Rest of SM figures. Small modifications of experiments in previous figures, specified explicitly in captions.

B Details of theoretical results

In this section we prove the main theoretical results. We begin with two technical lemmas that apply to k-homogeneous network functions, namely network functions $f_\theta(x)$ that obey the equation $f_{a\theta}(x) = a^k f_\theta(x)$ for any input x, parameter vector θ, and $a > 0$.

Lemma 1. *Let $f_\theta(x)$ be a k-homogeneous network function. Then $\sum_\mu \theta_\mu \partial_\mu \partial_{\nu_1} \cdots \partial_{\nu_m} f(x) = (k-m)\partial_{\nu_1} \cdots \partial_{\nu_m} f(x)$.*

Proof. We prove by induction on m. For $m = 0$, we differentiate the homogeneity equation with respect to a.

$$0 = \left.\frac{\partial}{\partial a}\right|_{a=1} \left(f_{a\theta}(x) - a^k f_\theta(x)\right) = \sum_\mu \frac{\partial f(x)}{\partial \theta_\mu}\theta_\mu - k f_\theta(x)\,. \tag{S1}$$

For $m > 0$,

$$\begin{aligned}\sum_\mu \theta_\mu \partial_\mu \partial_{\nu_1} \cdots \partial_{\nu_m} f(x) &= \partial_{\nu_m}\left(\sum_\mu \theta_\mu \partial_\mu \partial_{\nu_1} \cdots \partial_{\nu_{m-1}} f(x)\right) - \sum_\mu \left(\partial_{\nu_m}\theta_\mu\right)\partial_\mu \partial_{\nu_1} \cdots \partial_{\nu_{m-1}} f(x) \\ &= \partial_{\nu_m}(k-m+1)\partial_{\nu_1} \cdots \partial_{\nu_{m-1}} f(x) - \sum_\mu \delta_{\mu\nu_m}\partial_{\nu_m}\theta_\mu \partial_\mu \partial_{\nu_1} \cdots \partial_{\nu_{m-1}} f(x) \\ &= (k-m)\partial_{\nu_1} \cdots \partial_{\nu_{m-1}}\partial_{\nu_m} f(x)\,.\end{aligned} \tag{S2}$$

□

Lemma 2. *Consider a k-homogeneous network function $f_\theta(x)$, and a correlation function $C(x_1,\ldots,x_m)$ that involves derivative tensors of f_θ. Let $L=\sum_{x\in S}\ell(x)+\frac{1}{2}\lambda\|\theta\|_2^2$ be a loss function, where S is the training set and ℓ is the sample loss. We train the network using gradient flow on this loss function, where the update rule is $\frac{d\theta^\mu}{dt}=-\frac{dL}{d\theta^\mu}$. If the conjecture of Dyer and Gur-Ari [2020] holds, and if the conjecture implies that $C=\mathcal{O}(n^{-1})$ where n is the width, then $\frac{dC}{dt}=\mathcal{O}(n^{-1})$ as well.*

Proof. The cluster graph of C has m vertices; we denote by n_e (n_o) the number of even (odd) components in the graph (we refer the reader to Dyer and Gur-Ari [2020] for a definition of the cluster graph and other terminology used in this proof). By assumption, $n_e+(n_o-m)/2\leq -1$.

We can write the correlation function as $C(x_1,\ldots,x_m)=\sum_{\text{indices}}\mathbb{E}_\theta\left[\partial f(x_1)\cdots\partial f(x_m)\right]$, where $\mathbb{E}_\theta[\cdot]$ is a mean over initializations, $\partial f(x)$ is shorthand for a derivative tensor of the form $\partial_{\mu_1\ldots\mu_a}f(x):=\frac{\partial^a f(x)}{\partial\theta^{\nu^1}\ldots\partial\theta^{\nu^a}}$ for some a, and the sum is over all the free indices of the derivative tensors. Then $\frac{dC}{dt}=\sum_{b=1}^m C_b$, where $C_b:=\mathbb{E}_\theta\left[\partial f(x_1)\cdots\frac{d\partial f(x_b)}{dt}\cdots\partial f(x_m)\right]$. To bound the asymptotic behavior of dC/dt it is therefore enough to bound the asymptotics of each C_b.

Notice that each C_b is obtained from C by replacing a derivative tensor ∂f with $d(\partial f)/dt$ inside the expectation value. Let us see how this affects the cluster graph. For any derivative tensor $\partial_{\mu_1\ldots\mu_a}f(x):=\partial^a f(x)/\partial\theta^{\nu^1}\cdots\partial\theta^{\nu^a}$, we have

$$\begin{aligned}\frac{d}{dt}\partial_{\mu_1\ldots\mu_a}f(x)&=\sum_\nu\partial_{\mu_1\ldots\mu_a\nu}f(x)\frac{d\theta^\nu}{dt}\\&=-\sum_\nu\partial_{\mu_1\ldots\mu_a\nu}f(x)\left[\sum_{x'\in S}\partial_\nu f(x')\ell'(x')+\lambda\theta^\nu\right]\\&=-\sum_{\nu,x'}\partial_{\mu_1\ldots\mu_a\nu}f(x)\partial_\mu f(x')\ell'(x')-(k-a)\lambda\partial_{\mu_1\ldots\mu_a}f(x)\,.\end{aligned}\tag{S3}$$

In the last step we used lemma 1. We now compute how replacing the derivative tensor ∂f by each of the terms in the last line of (S3) affects the cluster graph, and specifically the combination $n_e+(n_o-m)/2$.

The second term is equal to the original derivative tensor up to an n-independent factor, and therefore does not change the asymptotic behavior. For the first term, the ℓ' factor leaves n_e and n_o-m invariant so it will not affect the asymptotic behavior. The additional $\partial_\mu f$ factor increases the number of vertices in the cluster graph by 1, namely it changes $m\mapsto m+1$. In addition, it increases the size of the graph component of $\partial_{\mu_1\ldots\mu_a}f(x)$ by 1, therefore either turning an even sized component into an odd sized one or vice versa. In terms of the number of components, it means we have $n_e\mapsto n_e\pm1, n_o\mapsto n_o\mp1$. Therefore, $n_e+(n_o-m)/2\mapsto n_e+(n_o-m)/2\pm1\mp\frac{1}{2}-\frac{1}{2}\leq n_e+(n_o-m)/2\leq 1$. Therefore, it follows from the conjecture that $C_b=\mathcal{O}(n^{-1})$ for all b, and then $dC/dt=\mathcal{O}(n^{-1})$. □

We now turn to the proof of Theorems 1 and 2.

Proof (Theorem 1). A straightforward calculation leads to the following gradient flow equations for the network function and kernel.

$$\frac{df_t(x)}{dt}=-\sum_{x'\in S}\Theta_t(x,x')\ell'(x')-k\lambda f_t(x)\,,\tag{S4}$$

$$\frac{d\Theta_t(x,x')}{dt}=-2(k-1)\lambda\Theta_t(x,x')+T_t(x,x')+T_t(x',x)\,,\tag{S5}$$

$$T_t(x,x')=-\sum_{x''\in S}\partial_{\mu\nu}f_t(x)\partial_\mu f_t(x')\partial_\nu f_t(x'')\ell'(x'')\,.\tag{S6}$$

Here $\ell'=d\ell/df$. In deriving these we used the gradient flow update $\frac{d\theta^\mu}{dt}=-\frac{\partial L}{\partial\theta^\mu}$ and Lemma 1. It was shown in Dyer and Gur-Ari [2020] that $\mathbb{E}_\theta[T_0]=\mathcal{O}(n^{-1})$. If then follows from Lemma 2 that $\mathbb{E}_\theta\left[\frac{d^mT_0}{dt^m}\right]=\mathcal{O}(n^{-1})$ for all m, where the expectation value is taken at initialization. Furthermore,

the results of Dyer and Gur-Ari [2020] imply that $\mathrm{Var}\left[\frac{d^m T_0}{dt^m}\right] = \mathcal{O}(n^{-2})$ and therefore $\frac{d^m T_0}{dt^m} \underset{p}{\to} 0$.[4] In the strict infinite width limit we can therefore neglect the T contribution in the following equation, and write

$$\frac{d^m \Theta_0(x,x')}{dt^m} = [-2(k-1)\lambda]^m \Theta_0(x,x')\,, \quad m = 0,1,\dots\,. \tag{S7}$$

The solution of this set of equations (labelled by m) is the same as for the any-time equation $\frac{d}{dt}\Theta_t(x,x') = -2(k-1)\lambda\Theta_t(x,x')$, and the solution is given by

$$\Theta_t(x,x') = e^{-2(k-1)\lambda t}\Theta_0(x,x')\,. \tag{S8}$$

□

Proof (Theorem 2). The evolution of the kernel eigenvalues, and the fact that its eigenvectors do not evolve, follow immediately from (2). The solution (3) can be verified directly by plugging it into (1) after projecting the equation on the eigenvector $\hat{e}_a$. Finally, the fact that the function decays to zero at late times can be seen from (3) as follows. From the assumption $k \geq 2$, notice that $\exp\left[-\frac{\gamma_a(t')}{2(k-1)\lambda} - (k-2)\lambda t'\right] \leq 1$ when $t' \geq 0$. Therefore, we can bound each mode as follows.

$$|f_a(x;t)| \leq e^{-k\lambda t}\left[e^{\frac{(\gamma_a(t)-\gamma_a)}{2(k-1)\lambda}}|f_a(x;0)| + |\gamma_a y| e^{\frac{\gamma_a(t)}{2(k-1)\lambda}}\int_0^t dt'\right]\,. \tag{S9}$$

Therefore, $\lim_{t\to\infty}|f_a(x;t)| = 0$. □

For completeness we now write down the solution (3) in functional form, for $x \in S$ in the training set.

$$\begin{aligned} f_t(x) = e^{-k\lambda t}\Bigg\{ &\sum_{x'\in S} \exp\left[\frac{\Theta_t - \Theta_0}{2(k-1)\lambda}\right](x,x')\, f_0(x') \\ &+ \sum_{x',x''\in S}\int_0^t dt'\, e^{-(k-2)\lambda t'} \exp\left[\frac{\Theta_t - \Theta_{t'}}{2(k-1)\lambda}\right](x,x')\,\Theta_0(x',x'')\,y(x'')\Bigg\}\,. \\ \Theta_t(x,x') = e^{-2(k-1)\lambda t}\Theta_0(x,x')\,. \end{aligned} \tag{S10}$$

Here, $\exp(\cdot)$ is a matrix exponential, and Θ_t is a matrix of size $N_{\mathrm{samp}} \times N_{\mathrm{samp}}$.

B.1 Deep linear fixed point analysis

Let's consider a deep linear model $f(x) = \beta W_L....W_0.x$, with $\beta = n^{-L/2}$ for NTK normalization and $\beta = 1$ for standard normalization. The gradient descent equation will be:

$$\Delta W^l_{ab} = -\eta\lambda W^l_{ab} - \eta\beta \vec{W}^{l+1}_a \overleftarrow{W}^{l-1}_{b\alpha} \sum_{(x,y)\in S} \ell'(x,y) x_\alpha \tag{S11}$$

where we defined:

$$\vec{W}^l \equiv W^L...W^l, \overleftarrow{W}^l_\alpha \equiv W^l...W^0_\alpha \tag{S12}$$

Evolution will stop when the fixed point ($\Delta W = 0$) is reached:

$$W^{L>l>0}_{ab} = \vec{W}^{l+1}_a \hat{W}^{l-1}_b; W^L_a = \hat{W}^{L-1}_a; W^0_{a\alpha} = \vec{W}^1_a \hat{W}^{-1}_\alpha \tag{S13}$$

$$\hat{W}^{l-1}_b \equiv -\frac{\beta}{\lambda}\overleftarrow{W}^{l-1}_{b\alpha} \sum_{(x,y)\in S} \ell'(x,y) x_\alpha; \overleftarrow{W}^{-1}_{a\alpha} = \delta_{a\alpha} \tag{S14}$$

Furthermore note that:

$$\vec{W}^l.\hat{W}^{l-1} = -\frac{1}{\lambda}\sum_{(x,y)\in S} \ell'(x,y) f(x) = \tilde{f} \tag{S15}$$

[4] See appendix D in Dyer and Gur-Ari [2020].

Now, we would like to show that, at the fixed point

$$\vec{W}_a^l = \tilde{f}^{L-l}\hat{W}_a^{l-1} \tag{S16}$$

This follows from induction:

$$\vec{W}^L = W^L = \hat{W}^{L-1} \tag{S17}$$

$$\vec{W}^l = \vec{W}^{l+1}(\vec{W}^{l+1}\hat{W}^{l-1}) = \tilde{f}^{L-l-1}\hat{W}^l(\vec{W}^{l+1}\hat{W}^{l-1}) = \tilde{f}^{L-l}\hat{W}^{l-1} \tag{S18}$$

Which has a trivial solution if $\tilde{f} = 0$. Let's assume that it is non-trivial. If we contract the previous equation with $\hat{W}^{l-1}$ we get:

$$\tilde{f} = \tilde{f}^{L-l}||\hat{W}^{l-1}||^2 \tag{S19}$$

We can finally set $l = 0$ and simplify:

$$\frac{\lambda^{L+1}}{\beta^2} = [-\sum_{(x,y)\in S} \ell'(x,y)f(x)]^{L-1} \sum_{(x,y),(x',y')\in S} \ell'(x,y)\ell'(x',y')x.x' \tag{S20}$$

At large n, to obtain a non-trivial fixed point $f(x)$ should be finite as $n \to \infty$. From the previous equation, this implies that $\frac{\lambda^{L+1}}{\beta^2} = \theta(n^0)$. In NTK normalization $\beta^2 = n^{-L}$, for $\lambda = \theta(n^{-\frac{L}{L+1}})$, we will get a non-trivial ($f(x) \neq 0$) fixed point. This also implies that these corrections will be important for $\lambda = \theta(n^0)$ in standard normalization since there $\beta = 1$. Note that if $\frac{\lambda^{L+1}}{\beta^2} \neq \Omega(n^0)$, we expect that we get the $\lambda = 0$ solution $\ell'(x,y) = 0$.

We can be very explicit if we consider $L = 1$ and one sample with $x, y = 1, 1$ for MSE loss. The fixed point has a logit:

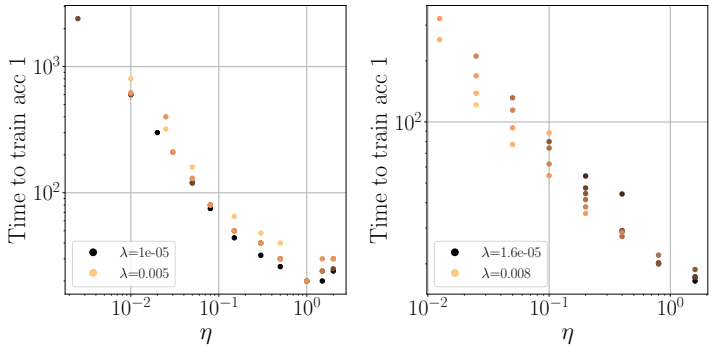
$$\lambda\sqrt{n} = f - 1 \tag{S21}$$

which is only different from $0, 1$ for fixed $\lambda^2 n$.

C More on experiments

C.1 Training accuracy = 1 scale

We can see how the time it takes to rech training accuracy 1 depends very mildly on λ, and for small enough learning rates it scales like $1/\eta$.

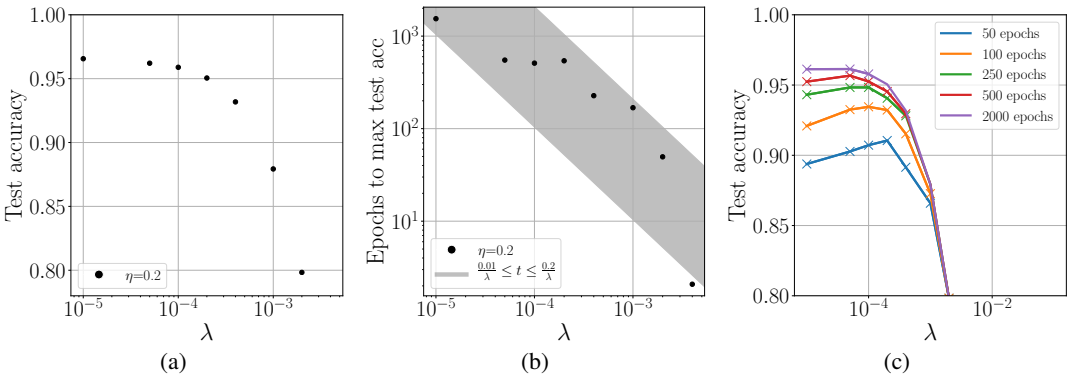


(a) FC (b) WRN

Figure S1: Training accuracy vs learning rate the setup of figure 2. The specific values for the η, λ sweeps are in A.

C.2 More WRN experiments

We can also study the previous in the presence of momentum and data augmentation. These are the experiments that we used in figure 4, evolved until convergence. As discussed before, in the presence of momentum the t_* depends on η, so we will fixed the learning rate $\eta = 0.2$.



(a) (b) (c)

Figure S2: WRN 28-10 with momentum and data augmentation trained with a fixed learning rate.

C.3 More on optimal L_2

Here we give more details about the optimal L_2 prediction of section 3. Figure S3 illustrates how performance changes as a function of λ for different time budgets with the predicted λ marked with a dashed line. If one wanted to be more precise, from figure 2 we see that while the scaling works across λ's, generally lower λ's have a scaling ~ 2 times higher than the larger λ's. One could try to get a more precise prediction by multiplying c by two, $c_{\text{small}\lambda} \sim 2c_{\text{large}\lambda}$, see figure S4. We reserve a more detailed analysis of this more fine-grained prescription for the future.

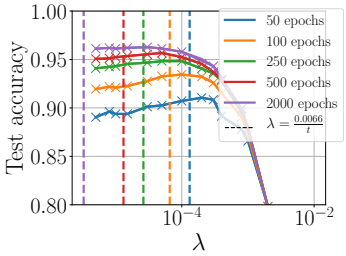


Figure S3: WRN trained with momentum and data augmentation. Given a number of epochs, we compare the maximum test accuracy as a function of L_2 and compare it with the smallest L_2 with the predicted one. We see that this gives us the optimal λ within an order of magnitude.

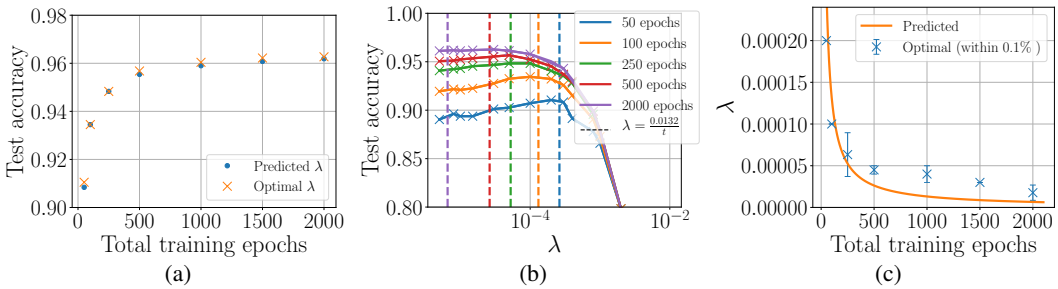


Figure S4: Same as previous figure with $c = 2c_{\text{large}\lambda} = 0.0132$.

C.4 MSE and the catapult effect

In Lewkowycz et al. [2020] it was argued that, in the absence of L_2 when training a network with SGD and MSE loss, high learning rates have a rather different final accuracy, due to the fact that at early times they undergo the "catapult effect". However, this seems to contradict with our story around 1.1 where we argue that performance doesn't depend strongly on η. In figure S5, we can see how, while when stopped at training accuracy 1, performance depends strongly on the learning rate, this is no longer the case in the presence of L_2 if we evolve it for t_{test}. We also show how the training MSE loss has a minimum after which it increases.

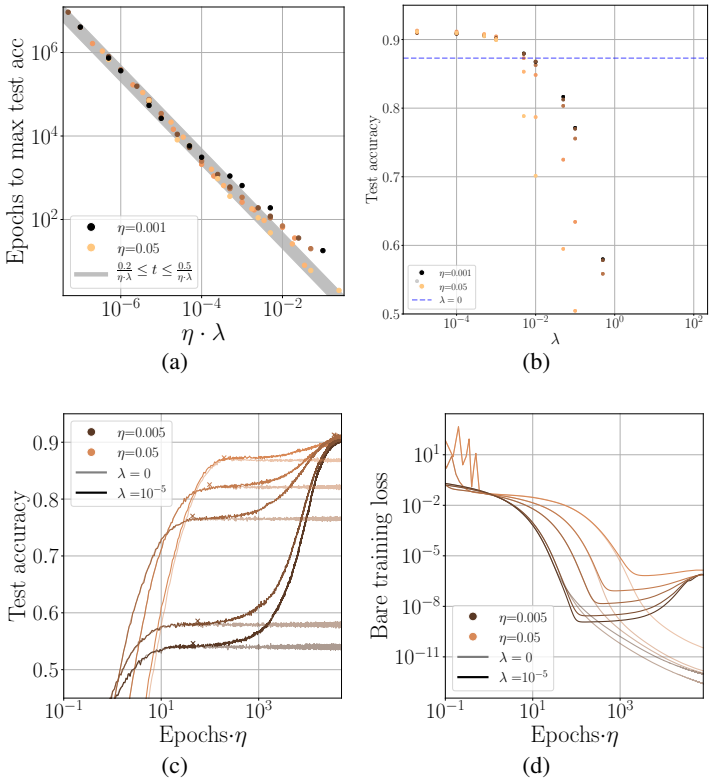
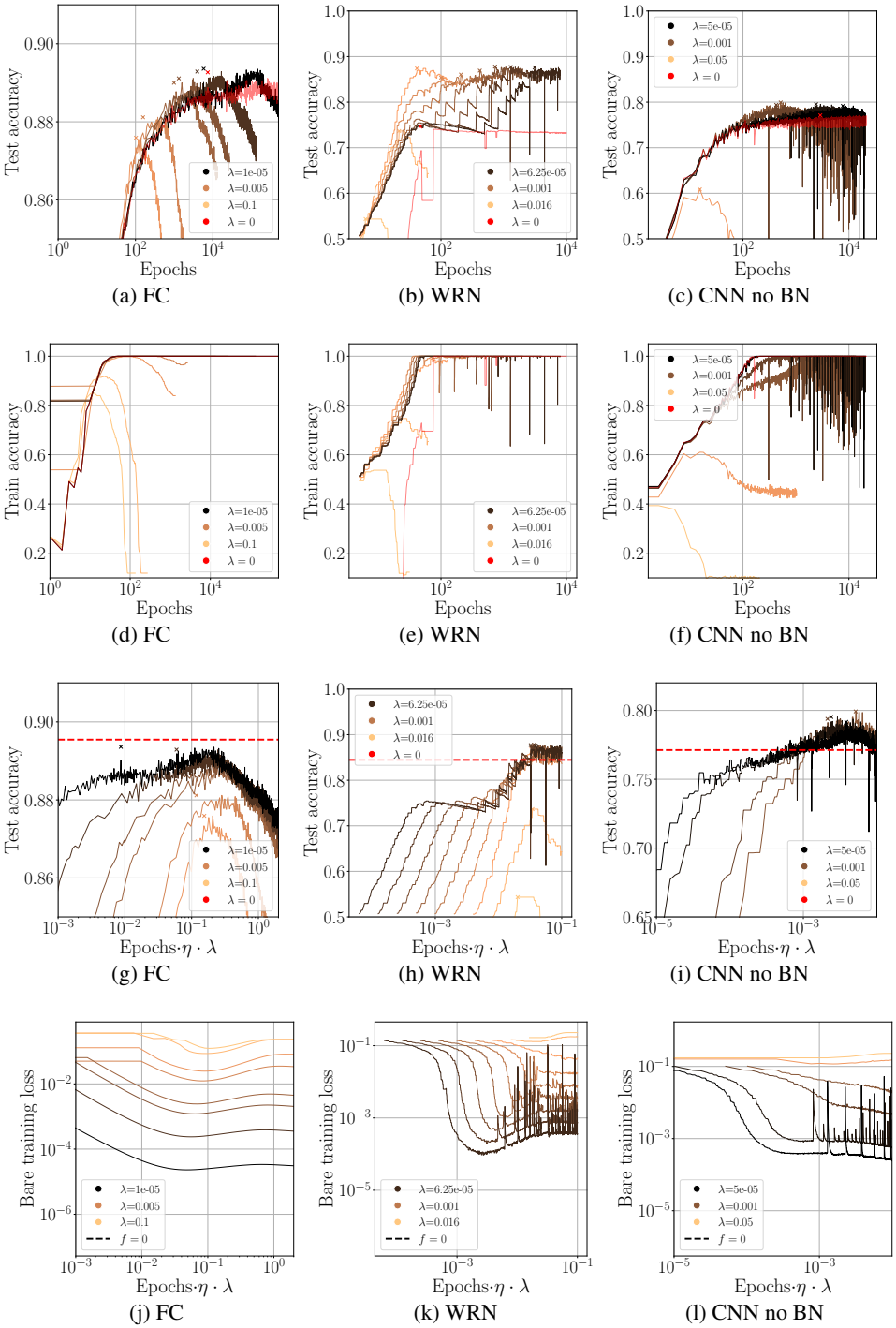


Figure S5: MSE and catapult effect: we see how even if there is a strong dependence of the test accuracy on the learning rate when the training accuracy is 1, this dependence flattens out when evolved until convergence in the presence of λ . The specific values for the η, λ sweeps are in A.

C.5 Dynamics of loss and accuracy

In figure S6 we illustrate the training curves of the experiments we have discussed in the main text and SM.



(a) FC (b) WRN (c) CNN no BN

(d) FC (e) WRN (f) CNN no BN

(g) FC (h) WRN (i) CNN no BN

(j) FC (k) WRN (l) CNN no BN

Figure S6: This shows the dynamics of experiments in figures 2, 3.The learning rates are 0.01 for the CNN and 0.2 for the WRN and 0.15 for the FC. Each legend has the min, max and median value of the logspace λ's with its respective color.

D Examples of setups where the scalings don't work

We will consider a couple of setups which don't exhibit the behaviour described in the main text: a 3 hidden layer, width 64 fully-connected network trained on CIFAR-10 and ResNet-50 trained on ImageNet. We attribute this difference to deviations from the overparametrized/large width regime. In this situation, the optimal test accuracy with respect to λ has a maximum at some $\lambda_{\text{opt}} \neq 0$.

For the FC experiment, the time it takes to reach this maximum accuracy scales like $1/\lambda$ for $\lambda \gtrsim \lambda_{opt}$, but becomes constant (equal to the value for $\lambda = 0$) for $\lambda \lesssim \lambda_{\text{opt}}$. This peak of the maximum test accuracy happens before the training accuracy reaches 1. Generically, we don't observe that a network trained with cross-entropy and without regularization to have a peak in the test accuracy at a finite time.

We do not have as clear an understanding of the ImageNet experimental results because they involve a learning rate schedule. Performance for small λs does not improve even if when evolving for a longer time. However, we do observe that performance is roughly constant when $\eta \cdot \lambda$ is held fixed.

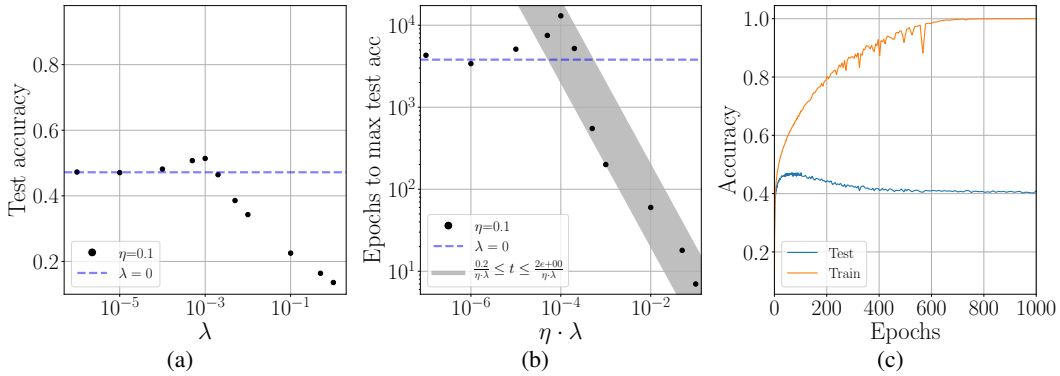


Figure S7: Experiments with FC of width 64 and depth 3 trained on CIFAR-10.

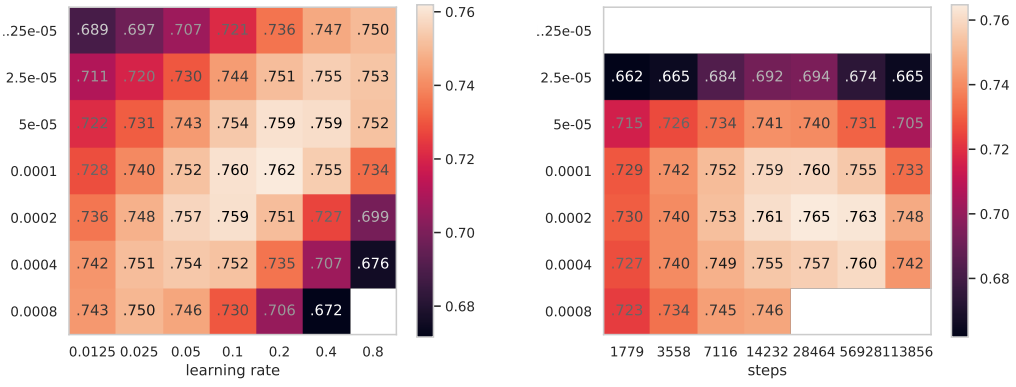


Figure S8: ResNet-50 trained on ImageNet. (a) Evolved for 14,232 epochs for different η, λ. While changing η or λ independently has a strong effect of performance, we see that performance is rather similar along the diagonal. (b) Fixed $\eta = 0.1$ and evolve for different λ and number of epochs T (rescaling the learning rate schedule to T). In constrast to the overparameterized case, we see that one cannot reach the same performance with a smaller λ by increasing T.

E More on AUTOL_2

The algorithm is the following:

Algorithm 1: AUTOL2

```
MINLOSS,MINERROR=∞, ∞
MIN_STEP,L2=0,0.1
for t in steps do
    UPDATE_WEIGHTS;
    if t mod k = 0 then
        MAKE_MEASUREMENTS;
        if ERROR_OR_LOSS_INCREASES AND t > MIN_STEP then
            L2=L2/10;
            MIN_STEP=0.1/L2 + t;
        else
            MINLOSS,MINERROR=min(LOSS_{t-k},MINLOSS),min(ERROR_{t-k},MINERROR);
Function ERROR_OR_LOSS_INCREASES (LOSS,ERROR,MINLOSS,MINERROR):
    if LOSS_t>MINLOSS AND LOSS_{t-k}>MINLOSS then
        return True;
    if ERROR_t>ERROR AND ERROR_{t-k}>ERROR then
        return True;
    return False;
```

We require the loss/error to be bigger than its minimum value two measurements in a row (we make measurements every 5 steps), we do this to make sure that this increase is not due to a fluctuation. After decaying, we force λ to stay constant for a time $0.1/\lambda$ steps, we choose the refractory period to scale with $1/\lambda$ because this is the physical scale of the system.

To complement the AUTOL_2 discussion of section 3 we have done another experiment where the learning rate is decayed using the schedule described in 2. Here we see how while AUTOL_2 trains faster in the beginning, the optimal $\lambda = 0.0005$ outperforms it. We have not hyperparameter tuned the possible parameters of AUTOL_2.

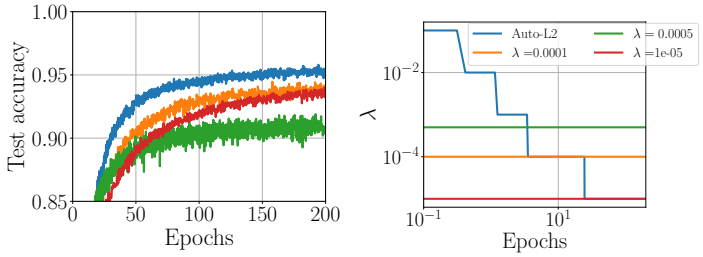


Figure S9: Here we have a WRN trained with momentum and data augmentation for 200 epochs. We compare the AUTOL_2 with different fixed L_2 parameters and we see how it trains faster and gets better accuracy.

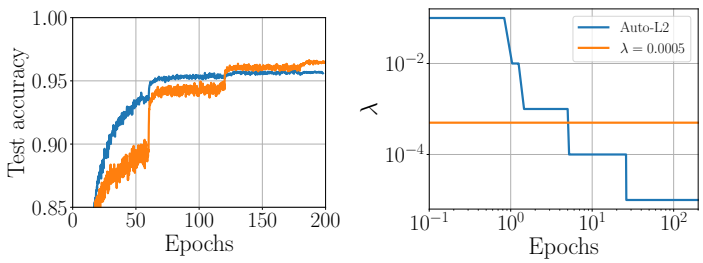


Figure S10: Setup of S9 in the presence of a learning rate schedule. While AUTOL_2 trains faster and better in the beginning, it can't keep pace with the big jumps of constant λ.