OPT2022: 14th Annual Workshop on Optimization for Machine Learning

Learning deep neural networks by iterative linearisation

Adrian Goldwaser AG2198@CAM.AC.UK
Hong Ge HG344@CAM.AC.UK
University of Cambridge

Abstract

The excellent real-world performance of deep neural networks has received increasing attention. Despite the capacity to overfit significantly, such large models work better than smaller ones. This phenomenon is often referred to as the scaling law by practitioners. It is of fundamental interest to study why the scaling law exists and how it avoids/controls overfitting. One approach has been looking at infinite width limits of neural networks (e.g., Neural Tangent Kernels, Gaussian Processes); however, in practise, these do not fully explain finite networks as their infinite counterparts do not learn features. Furthermore, the empirical kernel for finite networks (i.e., the inner product of feature vectors), changes significantly during training in contrast to infinite width networks. In this work we derive an iterative linearised training method. We justify iterative lineralisation as an interpolation between finite analogs of the infinite width regime, which do not learn features, and standard gradient descent training which does. We show some preliminary results where iterative linearised training works well, noting in particular how much feature learning is required to achieve comparable performance. We also provide novel insights into the training behaviour of neural networks.

1. Introduction

Deep neural networks perform well on a wide variety of tasks despite their overparameterisation and capacity to memorise random labels [12], often with improved generalisation behaviour as the number of parameters increases [10]. This goes contrary to classical beliefs around learning theory and overfitting, meaning there is likely some implicit regularisation inducing an inductive bias which encourages the networks to converge to well-generalising solutions. One approach to investigate this has been to examine infinite width limits of neural networks using the *Neural Tangent Kernel* (NTK) [4, 6], interestingly these often do worse than standard neural networks, though with extra tricks they can perform equivalently well or better under certain scenarios [7]. Similarly, despite their use for analysis due to having closed-form expressions for many terms, they don't predict finite network behaviour very closely in many regards. For example, due to not learning features, they cannot be used for transfer learning and the empirical NTK (outer product of Jacobians) changes significantly throughout training whereas NTK theory states in the infinite limit that this is constant. This raises important questions about in what ways they are different, most of which can be summarised as how the use of feature learning impacts the networks that can be learnt.

To work towards answering this question, we look at an interpolation between standard training and a finite analog of infinite training where we fix the empirical NTK by performing weight space linearisation. One interpretation of this is that we are varying the amount of feature learning allowed. We find that essentially any amount of feature learning is enough to eventually converge to a similar performing network, assuming learning rates are small enough.

1.1. Related Work

Li et al. [9] create an enhanced NTK for CIFAR10 with significantly better empirical performance than the standard one, however it still performs less well than the best neural networks. Yang and Hu [11] use a different limit to allow feature learning, however neither of these give much insight as to why the standard parameterisation doesn't work well.

Lee et al. [7] run an empirical study comparing finite and infinite networks under many scenarios and Fort et al. [3] look at how far SGD training is from fixed-NTK training, and at what point they tend to converge. Lewkowycz et al. [8] investigate at what points in training the kernel regime applies as a good model of finite network behaviour. Both find better agreement later in training.

Chizat et al. [2] consider a different way to make finite networks closer to their infinite width analogs by scaling in a particular way, finding that as they get closer to their infinite width analogs, they perform less well empirically.

2. Problem Formulation

Consider a neural network $f_\theta(x)$ parameterised by weights θ and a mean squared error loss function[1] $\mathcal{L}(\hat{Y}) = \frac{1}{2}||\hat{Y} - Y||^2$, where we minimise $\mathcal{L}\left(f_\theta(X)\right)$ for data X and labels Y. We can write the change in the function over time under gradient flow with learning rate η as:

$$\dot{\theta}_t = -\eta \nabla f_{\theta_t}(X)^\top (f_{\theta_t}(X) - Y) \tag{1}$$

$$\dot{f}_{\theta_t}(X) = -\eta \hat{\Theta}_t(X, X)(f_{\theta_t}(X) - Y) \qquad \text{where} \quad \left[\hat{\Theta}_t\right]_{ij} = \left\langle \frac{\partial f_{\theta_t}(X_i)}{\partial \theta}, \frac{\partial f_{\theta_t}(X_j)}{\partial \theta} \right\rangle \tag{2}$$

It has been shown [1, 4, 6] that in the infinite width limit the *empirical neural tangent kernel*, $\hat{\Theta}_t$, converges to a deterministic NTK, Θ. This is a matrix dependent only on architecture and data and does not change during training. From this perspective, training the infinite width model under gradient flow (or gradient descent with a small step size) is equivalent to training the weight-space linearisation of the neural network [6]. This raises a number of interesting observations about why this doesn't work well in finite networks and what is different in them. This is likely due to lack of enough random features whereas running gradient descent on the full network allows features to be learnt, reducing the reliance on having enough initial random features.

3. Iterative Linearisation

NTK theory says that if the width is large enough then training the weight-space linearisation is equivalent to training the full network [6]. However in practise training the fully linearised network performs very poorly for practically sized networks [7]. In this section we propose *iterative linearisation* in order to interpolate between training of the standard network and the linearised network.

Consider standard (full batch) gradient descent on a neural network.

$$\theta_{t+1} = \theta_t - \eta \phi_t (f_{\theta_t}(X) - Y) \qquad \text{where} \quad \phi_t = \nabla_\theta f_{\theta_t}(X)^\top$$

1. We use MSE for simplicity and compatability with NTK results here. While this is needed for some NTK results, it does not effect the algorithms we propose where any differentiable loss function can be used — see Appendix A

Here we can think of this as two separate variables we update each step, the weights θ_t and the features ϕ_t. However there is no requirement that we always update both, giving rise to the following generalised algorithm:

$$\theta_{t+1} = \theta_t - \eta\phi_s^{\text{lin}}\left(f_{s,t}^{\text{lin}}(X) - Y\right) \tag{3}$$

$$\phi_s^{\text{lin}} = \nabla_\theta f_{\theta_s}(X)^\top \tag{4}$$

where $s = K * \lfloor \frac{t}{K} \rfloor$. In addition, we write the linearised version of a neural network f_θ using its first order Taylor expansion at the weights θ_s as

$$f_{s,t}^{\text{lin}}(x) = f_{\theta_s}(x) + \nabla_\theta f_{\theta_s}(x)^\top(\theta_t - \theta_s)$$

Using this framework, when $K = 1$ this is simply gradient descent and when $K = \infty$ it is fully linearised training. Other values of K interpolate between these two extremes. See Algorithm 1 for more details. Note that we can also generalise this to not be periodic in terms of when we update ϕ so we call this *fixed period* iterative linearisation.

Algorithm 1: Iterative Linearisation (fixed period)

```
Input: learning rate $\eta$, update periodicity $K$, pre-initialised parameters $\theta_0$
for t = 1..epochs do
    $\theta_t \leftarrow \theta_{t-1} - \eta\nabla\mathcal{L}(f^{\text{lin}}(X))$;
    if t mod K = 0 then
        $f^{\text{lin}}(X) \leftarrow f_{\theta_t}(X) + \nabla f_{\theta_t}(X)^\top(\theta_t - \theta_0)$;
    end
end
```

3.1. Interpreting Iterative Linearisation

It is insightful to look a bit more closely at what is happening in Algorithm 1. In the case of standard training of a linear model, (e.g. $f^{\text{lin}}(\cdot)$), the 'features' are fixed and the learning decides how to use those features. This is what is happening in Equation (3) and with $K = \infty$, we don't learn any new features beyond the random ones we got through the initialisation of the network. Interestingly we can say the same for infinite width networks, this idea that infinite width networks don't learn features is not new (see Yang and Hu [11] for work trying to avoid this pitfall), but the finite analogy gives us a new perspective on what is happening in Algorithm 1. We then call Equation (4) *feature learning*, noting that the feature learning cannot be happening in Equation (3). From this interpretation, the Jacobian ϕ_t are the features we are using at time t and K tells us how frequently to update features, putting a limit on the frequency of feature learning updates.

4. Results

To examine the effect of increasing K, or equivalently reducing the feature learning frequency, we run a number of experiments on MNIST and CIFAR10 with a slightly larger variant of LeNet [5] with 50 channels in each convolutional layer and softmax output. The purpose of these experiments is not to achieve amazing performance on the datasets (it only gets to $\sim 50\%$ for CIFAR10) but

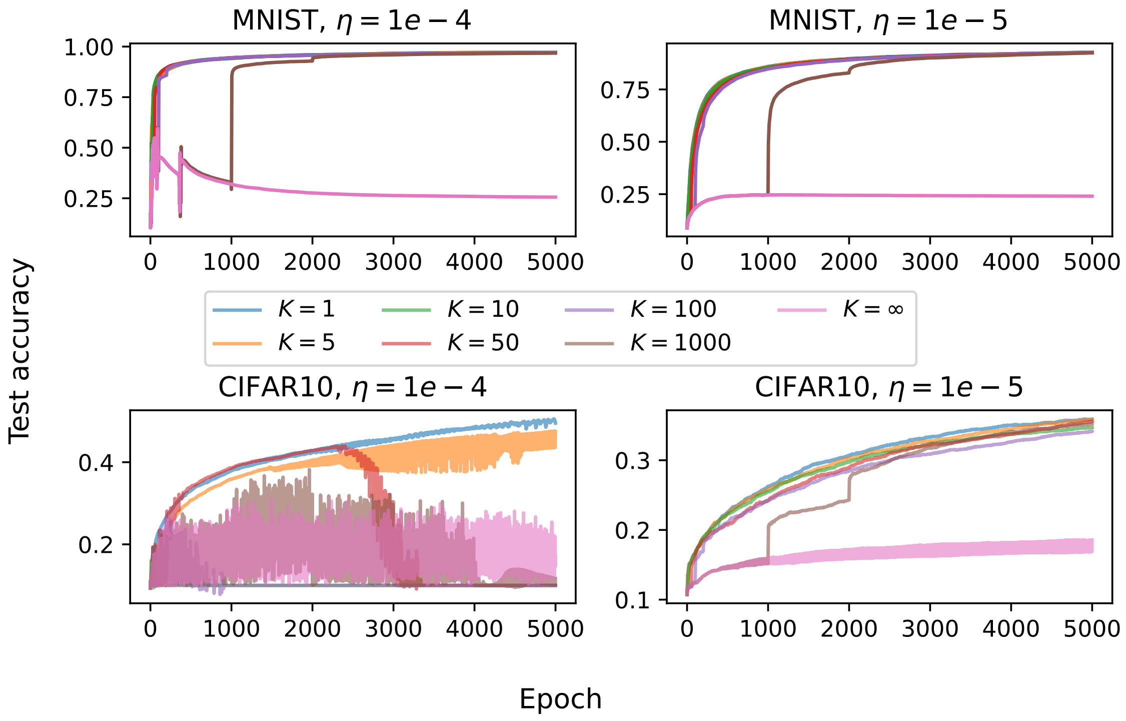

Figure 1: Iterative linearisation results on MNIST and CIFAR10 with learning rates of 1e-4 and 1e-5

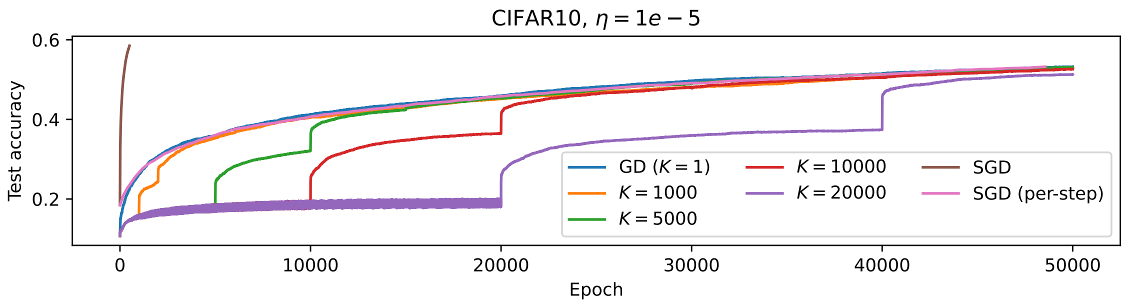

Figure 2: Iterative linearisation on CIFAR10 with large K

to examine how training changes as the learning rate η and update frequency K changes. For these experiments we note that unlike the general NTK theory, no process of our derivation relies on the use of MSE so we instead use cross-entropy loss as is standard for image classification. We additionally include the softmax in the loss function so it is not linearised by the algorithm, this both means that we continue to have an output for which taking the cross-entropy loss is meaningful as well as avoiding numerical issues caused by having the linearisation happening in the middle of the logsumexp trick.

Figure 1 shows the results of these experiments. For MNIST, all $K \leq 100$ follow almost the exact same learning trajectory, converging very quickly and not saturating the amount learnt from each set of features before updating. For $K = 1000$, it can be seen that the training levels off each time before ϕ is updated. This shows that the we can get close to 100% accuracy on MNIST by only updating the features from their initialisation twice (all except $K = \infty$ are within a 0.4% range at the end of training). From this we can conclude that the initialisation of a neural network with this architecture creates features that are not too far from what is needed to solve MNIST. Compare this with CIFAR10 (learning rate 1e-5) where it still only takes a few feature vector updates to reach the performance of $K = 1$ however this is a much lower accuracy. It is still unclear from these experiments how the few updates necessary interacts with different architectures on the same dataset.

Comparing CIFAR10 with learning rates of $1e-4$ and $1e-5$ also gives interesting conclusions. For $\eta = 1e - 4$, training diverges as soon as K goes above 5, and is unstable even for $K = 5$, whereas for $\eta = 1e - 5$ training stays stable for all K up to $K = 20000$ (see Figure 2). Note that this is not simply because it's moving less far as this is a 10x reduction in the step size but a 4000x increase in K. We include SGD training for comparison, noting that plotting per-step rather than per-epoch gives similar performance to GD training and so we should expect most runs to reach 60% accuracy if run for twice as long.

5. Conclusion

This paper has proposed *iterative linearisation*, a new training algorithm that interpolates between gradient descent on the standard and linearised neural network as a parallel infinite width vs finite networks. We show that, at least in the case of a LeNet-like architecture with small learning rates, any amount of features learning is enough to converge to a similar performing model. This provides an important step towards understanding feature learning and the distinction between how infinite and finite width networks learn. Better understanding how networks change with large amounts of parameters has important connections to empirical phenomena such as explaining deep double descent [10].

5.1. Future Work

It is important to do more rigourous empirical investigations to confirm these results, in particular to scale up to larger models in order to disentangle the impact of iterative linearisation training from the fact that this architecture will never do particularly well on CIFAR10. This is also important to better understand under what architectures/learning rates/frequencies iterative linearisation training is stable.

Another direction is to better understand the types of solutions that iterative linearisation finds for various values of K. This will shed light onto how the inductive bias is changing, in particular understanding if all $K < \infty$ find similar solutions and the infinite width limit is a step change similar to the test performance in the experiments here, or if this is a gradual change towards the solutions which don't learn features.

Finally, we only consider *fixed period* iterative linearisation here where we update the feature vector ϕ at regular intervals. However Fort et al. [3] showed that the empirical NTK changes faster earlier in training so it makes sense for K to be more adaptive if this was to be used directly for training.

References

[1] Sanjeev Arora, Simon S. Du, Wei Hu, Zhiyuan Li, Ruslan Salakhutdinov, and Ruosong Wang. On exact computation with an infinitely wide neural net. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 8139–8148, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/dbc4d84bfcfe2284ba11beffb853a8c4-Abstract.html.

[2] Lénaïc Chizat, Edouard Oyallon, and Francis R. Bach. On lazy training in differentiable programming. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 2933–2943, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/ae614c557843b1df326cb29c57225459-Abstract.html.

[3] Stanislav Fort, Gintare Karolina Dziugaite, Mansheej Paul, Sepideh Kharaghani, Daniel M. Roy, and Surya Ganguli. Deep learning versus kernel learning: an empirical study of loss landscape geometry and the time evolution of the neural tangent kernel. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/405075699f065e43581f27d67bb68478-Abstract.html.

[4] Arthur Jacot, Clément Hongler, and Franck Gabriel. Neural tangent kernel: Convergence and generalization in neural networks. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 8580–8589, 2018. URL https://proceedings.neurips.cc/paper/2018/hash/5a4be1fa34e62bb8a6ec6b91d2462f5a-Abstract.html.

[5] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proc. IEEE*, 86(11):2278–2324, 1998. doi: 10.1109/5.726791. URL https://doi.org/10.1109/5.726791.

[6] Jaehoon Lee, Lechao Xiao, Samuel S. Schoenholz, Yasaman Bahri, Roman Novak, Jascha Sohl-Dickstein, and Jeffrey Pennington. Wide neural networks of any depth evolve as linear models under gradient descent. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 8570–8581, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/0d1a9651497a38d8b1c3871c84528bd4-Abstract.html.

[7] Jaehoon Lee, Samuel S. Schoenholz, Jeffrey Pennington, Ben Adlam, Lechao Xiao, Roman Novak, and Jascha Sohl-Dickstein. Finite versus infinite neural networks: an empirical study. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/ad086f59924fffe0773f8d0ca22ea712-Abstract.html.

[8] Aitor Lewkowycz, Yasaman Bahri, Ethan Dyer, Jascha Sohl-Dickstein, and Guy Gur-Ari. The large learning rate phase of deep learning: the catapult mechanism. *CoRR*, abs/2003.02218, 2020. URL https://arxiv.org/abs/2003.02218.

[9] Zhiyuan Li, Ruosong Wang, Dingli Yu, Simon S. Du, Wei Hu, Ruslan Salakhutdinov, and Sanjeev Arora. Enhanced convolutional neural tangent kernels. *CoRR*, abs/1911.00809, 2019. URL http://arxiv.org/abs/1911.00809.

[10] Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. Deep double descent: Where bigger models and more data hurt. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=B1g5sA4twr.

[11] Greg Yang and Edward J. Hu. Tensor programs IV: feature learning in infinite-width neural networks. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 11727–11737. PMLR, 2021. URL http://proceedings.mlr.press/v139/yang21c.html.

[12] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=Sy8gdB9xx.

Appendix A. Iterative linearisation with a general loss function

In Section 3 we show how to get to iterative linearisation from standard gradient under mean squared error loss. The use of mean squared error is more instructive due to its similarities with NTK results, however it is not strictly necessary. For completeness we include here the same idea but for a general loss function $\mathcal{L}(\cdot)$.

Standard gradient descent on a function $f_\theta(\cdot)$ parameterised by θ, with step size η and data X can be written as

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta \mathcal{L}(f_{\theta_t}(X))$$

We can apply the chain rule, resulting in

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta f_{\theta_t}(X) \mathcal{L}'(f_{\theta_t}(X))$$

Where $\mathcal{L}'(\cdot)$ is the derivative of $\mathcal{L}(\cdot)$ (in the case of mean squared error, this is the residual: $\mathcal{L}'(\hat{Y}) = \hat{Y} - Y$). Now again using $\phi_t = \nabla_\theta f_{\theta_t}(X)$, we can write this as

$$\theta_{t+1} = \theta_t - \eta \phi_t \mathcal{L}'(f_{\theta_t}(X))$$

With a similar argument to Section 3, we note that we don't need to update the features ϕ_t every step, resulting in the following formulation.

$$\theta_{t+1} = \theta_t - \eta \phi_s^{\text{lin}} \mathcal{L}' \left(f_{s,t}^{\text{lin}}(X) - Y \right) \tag{5}$$

$$\phi_s^{\text{lin}} = \nabla_\theta f_{\theta_s}(X) \tag{6}$$

where $s = K * \lfloor \frac{t}{K} \rfloor$

This now lets us use softmax followed by cross-entropy in the loss $\mathcal{L}(\cdot)$ while maintaining the same interpretation, as we do for the MNIST and CIFAR10 results.