ATML Report

Ashutosh Singh
`ashutosh.singh@usi.ch`

Burgunder, Michal
`michal.burgunder@usi.ch`

Uslu, Muhammet
`muhammet.uslu@usi.ch`

Abstract

The training and eventual accuracy of machine learning models is a vital part to the discipline. A previous paper postulated that during training, a two misleading descents can be observed, and suggested that this arose from an interplay between the biases and variances' of training data, and made the suggestion to modify the step size in accordance to these values. This paper attempts to reproduce the results given in that paper on the 3 models tested on: A Linear model, a 2-Layer Neural Network and a 5-layer convolutional neural network and ResNet-18 model

1 Introduction

Test errors in of Machine Learning algorithms have a double descent characteristic; where error decreases first w.r.t a metric, then increases and then again decreases as the metric is increased. Two such metrics of great importance are *model-size* and *training-time*.

In [1] the authors suggest a method to deal with the double descent in test error of different machine learning models. They deal with the double-descent w.r.t *training-time* or *epochs* as observed by [2]. Understanding this *epoch-wise double-descent* is very important for determining the optimal *Early Stopping* time. The authors experiment with a Linear Model, Two-Layer 5-layer Convolutional Network and ResNet-18.[3].
We were fairly successful in using code provided by original authors to reproduce their results. We fixed the code at a number of places for it work we also added code to re-produce the results for *ResNet* architecture which were absent from the original code.

2 Related works

Each step of an iterative algorithm reduces the bias but increases variance. Thus early stopping can ensure that neither bias nor variance are too large. A variety of papers [4]; [5]; [6]; [7] formalized this intuition and developed theoretically sound early stopping rules. Those works predict that a double descent curve can occur. For a linear least squares problem, the data in the direction of singular vectors associated with large singular values is fitted faster than that in the direction of singular vectors associated with small singular values. [8] have shown this for a linear least squares problem or stated differently, a linear neural network with a single layer. [9] and [10] have shown that this view explains why neural network often fit clean labels before noisy ones, [11] have used this view to prove that convolutional neural networks provably denoise images.

3 Methodology

3.1 Early stoppıng in linear least squares

The risk as a function of the early stopping time is characterized by a superposition of U-shaped bias-variance tradeoffs, and if the features of the Gaussian linear model have different scales, those bias-variance tradeoff curves add up to a double descent shaped risk curve. We also show that the early stopping performance of the estimator can be improved through double descent elimination by scaling the stepsizes associated with the features.
To find the risk associated with some data set, we apply the formula as given in the original paper:

$$\bar{R}(\tilde{\theta}^t) := \sigma^2 + \sum_{i=1}^{d} \underbrace{\sigma_i^2(\theta_i^*)^2(1-\eta_i\sigma_i^2)^{2t} + \frac{\sigma^2}{n}(1-(1-\eta_i\sigma_i^2)^t)^2}_{U_i(t)}$$

where σ is the standard deviation of the entire data set, σ_i is the standard deviation of a given data batch, η_i is the stepsize for a given batch and t is the iteration.

The least squares method is given by the below:

$$|R(\theta^t) - \bar{R}(\tilde{\theta}^t)| \leq c\Big(\frac{\max_i \eta_i^2 \sigma_i^4}{\min_i \eta_i \sigma_i^4} \frac{d}{n} \Big(||\Sigma\theta^*||_2^2 + \frac{d}{n}\sigma^2 \log(d)\Big) \frac{\sigma^2}{n}\sqrt{d}\Big)$$

Here, c is a numerical constant, where θ^* and Σ are parameters of Linear Gaussian.
The risk of early stopped least-squares is a superposition of U-shaped bias variance trade offs, and if the features are differently scaled, this can give rise to epoch-wise double descent.

3.2 Early stopping in two layer neural networks

Double descent is eliminated by choosing a smaller stepsize for the second layer, or by choosing a smaller initialization for the first layer. Also note that,it also gives a better overall risk.

The first step in explaining the methodology is defining the neural net as original authors describe it. Thw two layer network model with ReLU can be given as -

$$f_{\mathbf{W}_t\mathbf{v}_t}(\mathbf{x}) = \frac{1}{k} ReLU(\mathbf{x}^T\mathbf{W})\mathbf{v} \tag{1}$$

here $\mathbf{x} \in \mathbb{R}^d$ is the input generated as described in the Data Model, $\mathbf{W} \in \mathbb{R}^{d\times k}$, the weights of first layer and $\mathbf{v} \in \mathbb{R}^k$ the weights of the second layer.

Data Model: The dataset used to train this model is sampled from a joint distribution. The supervised training set

$$\mathcal{D} = \{(x_1, y_1)...(x_n, y_n)\}$$

where each example (x_i, y_i) is sampled from the joint distribution.The datapoints are normalized i.e. $\|x_i\|_2 = 1$ and the labels are bounded i,e $y_i \leq 1$.

The network is trained on an early-stopped, randomly initialized gradient descent with quadratic loss.

$$[\mathbf{W}^0]_{ij} = \mathcal{N}(0, \omega^2)$$
$$[\mathbf{v}^0]_i = Uniform(\{-\nu, \nu\})$$

Here, ω and ν are parameters that trade off the magnitude of the weights of the first and second layer. Note that with this initialization, for a fixed unit norm feature vector $\mathbf{x}$, we have $f_{\mathbf{W}_0,\mathbf{v}_0}(x) = O(\nu\omega)$. We apply gradient descent to the mean-squared loss.

$$\mathcal{L}(\mathbf{W}, \mathbf{v}) = \frac{1}{2}\sum_{i=0}^{n}(y_i - f_{\mathbf{W}\mathbf{v}}(\mathrm{x}_i^2)) \tag{2}$$

The test error or risk for the network is defined by the original authors[12] as

$$R(f) = \mathbb{E}[l(f(\mathbf{x}), y)] \tag{3}$$

where $(\mathbf{x}, y)$ are the training pairs and $l : [\mathbb{R} \times \mathbb{R}] \to [0, 1]$ is the loss function; which is is 1-Lipschitz in it first argument and obeys $l(y, y) = 0$

Now that the network is defined the idea here is to give a bound on the test error of the two layer neural net as described above and use that to study the the epoch-wise double descent. This lower bound us give in terms of a Gram Matrix by the original authors[12] associated with the two kernels of the first and second layer of the network.

$$\Sigma_{ij} = \nu^2 K_1(x_i, x_j) + \omega^2 K_2(x_i, x_j) \tag{4}$$

where x_i and x_j are the training examples. With kernels

$$K_1(x_i, x_j) = \frac{1}{2}\left(1 - \frac{\cos^{-1}(\rho_i j)}{\pi}\right)\rho_{ij}$$

$$K_2(x_i, x_j) = \frac{1}{2}\left(\frac{\sqrt{1-\rho_{ij}^2}}{\pi} - \frac{\cos^{-1}(\rho_i j)}{\pi}\right)\rho_{ij}$$

where $p_{ij} = \langle x_i, x_j \rangle$.

The main result is given by the SVD of this Gram Matrix.

$$\Sigma = \sum_{i=1}^{N} \sigma_i^2 \mathbf{u}_i \mathbf{u}_i^T \tag{5}$$

The risk bound with a probability $1-\delta$ as given by Theorem-2[12] of a trained network for t iterations by gradient descent.

$$R(f_{\mathbf{W}_t}\mathbf{v}_t) \leq \sqrt{\frac{1}{n}\sum_{i=1}^{n}\langle u_i y\rangle^2(1-\eta\sigma_i^2)^{2t}} + \sqrt{\frac{1}{n}\sum_{i=1}^{n}\langle u_i y\rangle^2\left(1-\frac{(1-\eta\sigma_i^2)^t}{\sigma_i^2}\right)^2} + O\left(\frac{1}{\sqrt{n}}\right) \tag{6}$$

where $\alpha > 0$ is the smallest eigen value of *Gram Matrix*; and $k > \Omega\left(\frac{\eta^{10}}{\alpha^{15} min(\omega, \nu)}\right)$ where k is the width of the network and initialization scale paramteres obey $\nu\omega \leq \frac{alpha}{\sqrt{32\log\left(\frac{2n}{\delta}\right)}}$ and $\nu + \omega \leq 1$ for some $\delta \in (0,1)$

The risk bound established in 6 can be interpreted as a superposition of n bias-variance trade-off curves. Where the $i-th$ bias term decreases in the number of iterations and $i-th$ Whether epoch-wise double descent occurs or not depends on those singular values and therefore on the kernels, the initialization, and the distribution of the examples. term increases in the number of iterations. The speed at which these two terms increase or decrease depends on the $i-th$ Singular value σ_i of the $GramMatrix$ which depends upon the kernels K_1 and K_2, the random initialization(specifically ν and ω; the scale parameters) and the training examples.

To explain upscaling or downscaling of the weights or learning rates the authors[12] give a different view of the above established idea of when double descent happens. They capture the relationship between singular values of Σ and parameters of first and second layer by approximating the NN with its Linear approximation around the initialization. With this approximation, the networks predictions for the n training examples are approximately given by -

$$\begin{bmatrix} f_{\mathbf{W}_t}\mathbf{v}_t(x_1) \\ \vdots \\ f_{\mathbf{W}_t}\mathbf{v}_t(x_n) \end{bmatrix} \approx \mathbf{J}\begin{bmatrix} Vect(\mathbf{W}) \\ \mathbf{v} \end{bmatrix} = \sum_{i=0}^{n} \sigma_i \mathbf{u}_i((\mathbf{v}_{i\mathbf{W}}^T Vect(\mathbf{W}) + \mathbf{v}_{i\mathbf{v}}^T \mathbf{v}) \tag{7}$$

where $\mathbf{J} \in \mathbb{R}^{n\times dk+k}$ is the approx Jacobian of the network at initialization and $J = \sum_{i=1}^{n}\sigma_i^2 \mathbf{u}_i \mathbf{v}_i^T$ is its SVD. $\mathbf{v}_{i\mathbf{W}} \in \mathbb{R}^{dk}$ and $\mathbf{v}_{i\mathbf{v}} \in \mathbb{R}^{k}$ are the parts of right singular vectors of $\mathbf{J}$ associated with parameters of first and second layer respectively.

The norm of the vectors $\mathbf{v}_{i\mathbf{W}}$ and $\mathbf{v}_{i\mathbf{W}}$ measures to what extent the singular value σ_i is associated with the weights in the first and second layer. If norm of $\mathbf{v}_{i\mathbf{v}}$ is larger for large singular values then this layer learns faster; bias term decreases quickly and variance term increases quickly for first layer so scaling down the learning rate to make smaller updates *or* scaling up the the weights(using the weight scaling param ν) to make updates smaller in comparision to original value(slower learning), we can control the double descent w.r.t training time.

3.3 Early stopping in Deep Neural networks

Epoch-wise double descent can be eliminated and the early stopping performance can be improved by adjusting the stepsizes/learning rates. Since large singular values are associated mostly with the weights of the fully connected layers. This causes the convolutional layers to be learned slower than the fully connected layer which results in double descent. So Whether epoch-wise double descent occurs or not depends on those singular values and therefore on the kernels, the initialization, and the distribution of the examples.

5-Layer CNN(MCNN)

Double descent can be eliminated by changing the stepsize of the final layer, this time by decreasing the stepsize of the final fully connected layer. This scaling of the learning rate of last layer is based on the idea that the larger singular values of the *Jacobian* are associated with mostly the *Fully Connected Layer* causing it to learn at a much faster rate then the *Convolutional Layers*. So scaling the learning rate to a lower value solves the problem of double-descent in test error w.r.t training time.

ResNet-18

the double descent behavior occurs because some layer(s) of the ResNet-18 model are fitted at a faster rate than others, then scaling the learning rates of some layers should eliminate double descent.

4 Implementation

We re-use the original authors' code to reproduce the results. We had to modify and re-implement the original code at number of places to get it into running condition.
We started with the writing a script to transform CIFAR-10 dataset to the format required by the original authors' code. We are providing this script in the new code. We also have implemented the capability to to train ResNet-18; which was a part of original results, but was not provided in to code.

In addition to that, for training MCNN and ResNet we correct the original code at multiple places, for eg. handling *DataParallel(PyTorch [13])*, checkpoint saving code and other places. We used a node on the CSCS (Swiss National Supercomputing Centre) with two GeForce RTX 2080 Ti GPUs.

We also use the original authors' code with few correction to reproduce results of Two-Layer NN. Details in the following sections

For the linear model, we have taken the existing code base, and modified four parameters: σ_1, σ_2, θ_1 and θ_2. We have modified these parameters only slightly. Each value change has been measured against every other value change, according to a step size, as well as a starting and ending value. See table below.

For sigma1, we modified the value by 0.05 in for starting from 0.85 to 1.1. In order to give a bit of flexibility to our model, we are assigning a threshold value of 5, as the maximal difference between the predicted location of the minimum, and the actual location of the minimum, to account for changes in the hyper parameters.

parameter	step size	start value	end value
σ_1	0.05	0.85	1.1
σ_2	0.01	0.125	0.130
θ_1	0.1	1.25	1.30
θ_2	0.1	0.8	1.3

4.1 Datasets

In this section we describe all the datasets used by original authors and then by us to recreate the original results

Linear Model

Given that the data for the linear model has been generated from random distributions, we have used the same data creation mechanism to recreate the data as used for the paper. For this, the standard deviations, thetas, s.d. noise have been set manually. To create new data sets, these base values have been modified to generate new data sets.

Two-Layer NN

The dataset used to train this model is sampled from a joint distribution. The supervised training set

$$\mathcal{D} = \{(x_1, y_1)...(x_n, y_n)\}$$

where each example (x_i, y_i) is sampled from the joint distribution.The datapoints are normalized i.e. $\|x_i\|_2 = 1$ and the labels are bounded i,e $y_i \leq 1$.

We use the settings provided by original authors to generate from a mixture of Standard Normal Distributions. We use the Sigma Ranges and code provided by authors to generate the dataset.

MCNN

We use noisy CIFAR-10[14] dataset for training the 5-layer Convolutional Neural Network. **20%** label noise was introduced in the original CIFAR-10 dataset. We also transform the dataset from raw format to .npz format as required by the code of original authors.

ResNet-18

We use noisy CIFAR-10[14] (10 class classification) dataset, since the risk (or test error) only has a double descent behavior if the network is trained on a dataset with label noise we trained with **20%** random label noise similar to MCNN to train and test the ResNet-18 model.

4.2 Hyperparameters

Linear Model

The values mentioned above (σ_1, σ_2, θ_1, θ_1) have initially been individually tweaked based on guessing, to see where the limits are of what the paper is claiming. the $kmax$ parameter has also been decreased from $10'000$ to $1'000$. The other hyperparameters have been left as given. The changes in values have been given in the Implementation section.

Two-Layer NN

We also experiment with the learning rate fixed but weights of the individual layers scaled with params ω and ν. We test three settings $\omega = 0.1; \nu = 1$, $\omega = 1; \nu = 0.1$ and $\omega = 1; \nu = 1$ For each of the above setting of params we do two runs for this network. Once with same learning rate with value ***0.00008*** for both the layers and again with learning rates ***0.000001*** for second and ***0.00008*** for first layer respectively respectively. All these numbers are provided by original authors.

We train for *50k* iterations. We generated the dataset from a joint distribution of standard normal distributions defined by a sigma range. We use the sigma range *1 - 4* as used by the original authors.

MCNN

We do three runs for the network once with same learning rate, again with learning rate for last linear layer scaled down and learning rate of thje same layer scaled up

We initialize each layer with a learning rate of ***0.1*** or ***0.05***. We scale the learning rate of last layer by a scaling factor of ***0.001*** for the second run scale it up in the third run with learning rate of ***0.2***
We use the SGD (Stochastic Gradient Descent) optimizer with learning rate ***0.1***, we use Inverse Square Root LR[use reference] with a inverse decay rate of ***512.0***.We use SGD for ***0.0*** weight Decay and ***0.0*** momentum just like original authors.

ResNet-18

We do two runs for the network once with same learning rate and again with learning rate for the second half of the network scaled down

We initialize each layer with a learning rate of ***0.1***. We scale down the weights of the last three layer groups by ***0.05***, ***0.02*** and ***0.001*** respectively.
To define loss function (criterion) and optimizer We use the SGD optimizer with learning rate ***0.1***, we use Inverse Square Root LR[use reference] with a inverse decay rate of ***512.0***.We use SGD for ***0.0*** weight Decay and ***0.0*** momentum just like original authors, the loss funtion is Cross Entropy loss from PyTorch modules.

4.3 Experimental setup

More details on exact commands and setup can be found in README.md of this git-respository

Linear Model

Given that the linear model requires relatively little computation, the model was simply run on the laptop that was available. All of the code was written in python 3, using a 2.6 GHz Intel Core CPU.

MCNN

The implemented code is in PyTorch . We train and record test errors of the 5-layer Convolutional Neural Network two times, once with learning rate decay and again with lower learning rate for the last Linear layer. We used GeForce RTX 2080 Ti for training the network for *2k* epochs each time using SGD optimizer. In the second run we have the used the scaling factor of ***0.001*** provided by original authors for the last linear layer of the network.
We use a GPU node on the PizDaint SuperComputer at CSCS to train the model. Using the *slurm* command ***sbatch*** we train the model on two GPUs.

Two-Layer NN

We used the original notebook provided by original authors' and modified it to be able to recreate all the resuts with different hyperparam settings.

ResNet-18

Comparison to other models in the paper ResNest-18 was more complex network, using regular hardware or open source Google Collab [ref] was not sufficient to train such a deep model. We used GeForce RTX 2080 Ti for training the network for *2k* epochs each time using SGD optimizer.

4.4 Computational requirements

Provide information on computational requirements for each of your experiments. For example, the number of CPU/GPU hours and memory requirements. You'll need to think about this ahead of time, and write your code in a way that captures this information so you can later add it to this section.

Linear Model

No computational requirements for the Linear model could be identified.

Two-Layer NN

No specific computational requirements. We were able to train and test it pretty quickly using just the CPU on Google Collab with 2 Intel(R) Xeon(R) CPU @ 2.30GHz processors.

MCNN

We had to train the 5-layer Convolutional Neural Network for ***6*** hours with two GeForce RTX 2080 Ti GPUs to complete *2K* epochs. Each epoch includes validation and test phases as well. On each GPU the training takes up around *1500MB* space.
We have not optimized the code for better performance, we run the code as is from the original authors however we have modified the code at few places to make it run on multiple GPUs parallelly.

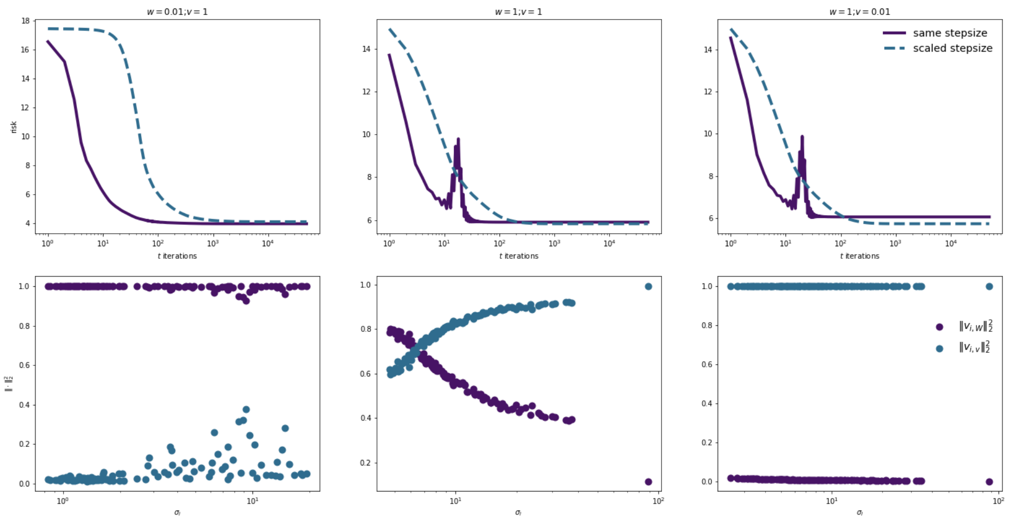
ResNet-18

We had to train the ResNet-18 for ***10*** hours with two GeForce RTX 2080 Ti GPUs to complete *2K* epochs. Each epoch includes validation and test phases as well. On each GPU the training takes up around *1700MB* space.
We added the ResNet-18 model to the code given by the original authors and the modifications to made by us to train the model on multiple GPUs in parallel.

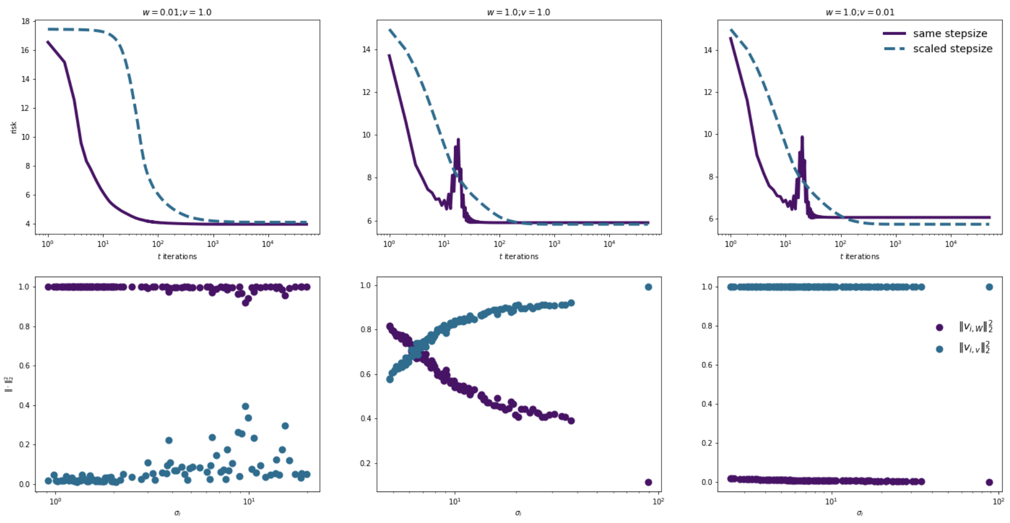
5 Results

For the linear model, slight variations in σ_1, σ_2, θ_1 and θ_2 have still produced accurate readings of the double descent model, given a slight threshold. Of the 625 different variations in hyperparameters, only 10 of these do not land within the error brackets defined. For this reason, the double descent hypothesis given by the original authors is mostly satisfied (98.4%) for the linear model.

We evaluate the two-layer neural net with different weight scaling factors and learning rate scaling(for details see Hyperparameters section). In Figure-1 we present the results from our iterations and original authors and compare the two.



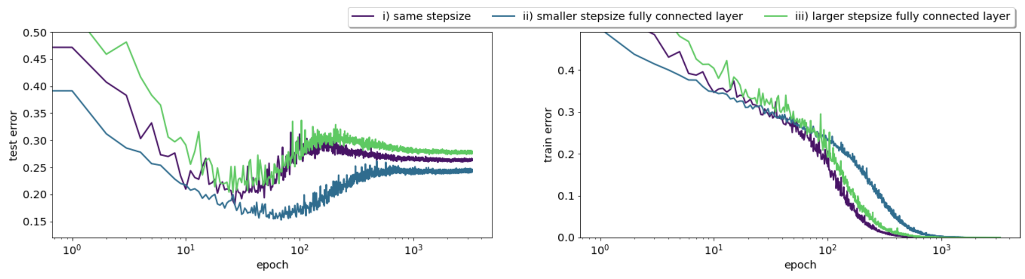
(a) Results from Original Authors



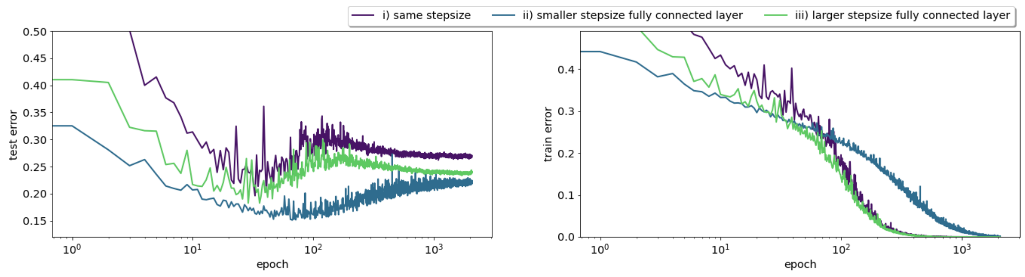
(b) Reproduced Results

Figure 1: Results for Two-Layer NN by original authors and from our reproducability experiment

It is clear from Figure-2 that we were able to reproduce the original results after fixing the code of original authors. The slight variations are only vertical shifts suggesting randomness in initialization or some slight changes in hyper-parameters from when they were used by original; we don't change them but maybe the hyper-parameters used by original authors were slightly different from what they released with code.
As inspired by author's theory the double descent behavior occurs because some layer(s) of the ResNet-18 model are fitted at a faster rate than others. Figure-3 shows that when scaling the step sizes of the later half of the layers of the network mitigates double descent.

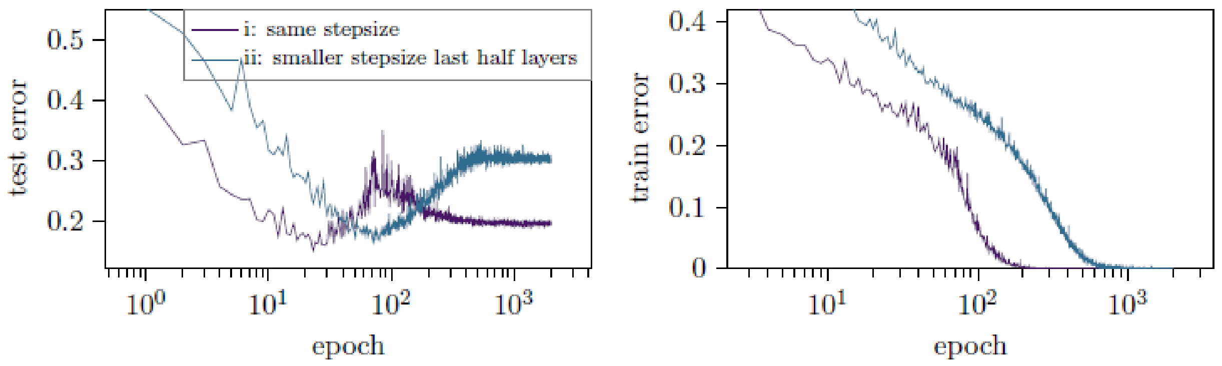


(a) Results from Original Authors

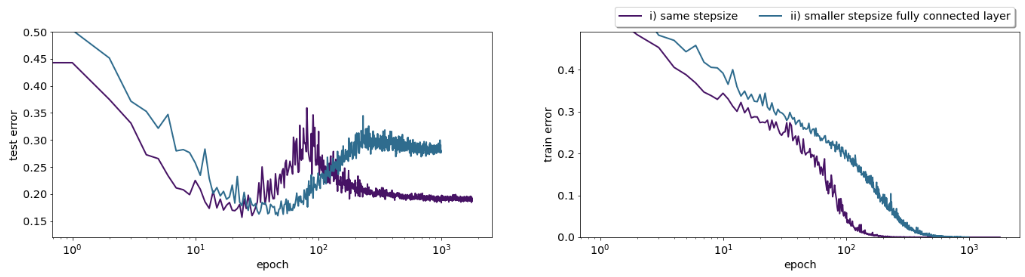


(b) Reproduced Results

Figure 2: Test and Train times for 5-Layer CNN. Each plot has three curves 1. Same stepsize; 2. Smaller stepsize for FC Layer; 3. Larger step size for FC layer. **a**: Results from three runs by original authors **b**: Results after reproducability experiments.



(a) Results from Original Authors



(b) Reproduced Results

Figure 3: **Left**: Test error of the ResNet-18 trained with the i) same step size for all layer, and with ii) a smaller step size for the latter half of the layers. Decreasing the learning rate of the last layers causes the last layers to be learned at a similar speed as the first and thereby eliminates double descent. **Right** : The training error curves for i) and ii). **a**: Results from three runs by original authors **b**: Results after reproducability experiments

6 Discussion and conclusion

Given the results of the paper on which this paper is based on, we were able to recreate the misleading double descent problems for Linear Least Squares, a Two Layer neural network and a MCNN. The suggested fix for each of these models has also been applied during training and analysis, and we confirm that the fixes presented are correct.
While identifying the problem is an important step to building more effective machine learning pipelines, implementing this fix for any model might be a challenge, given that current machine learning pipelines do not modify stepsize by default.

While reproducing the code we had to fix a number of errors in the scripts by original authors' so we think that reproduction of research code can be more easier with a final test run when publishing the code. We even found the code to be incomplete for reproducing the results where a complete model(ResNet-18) and all its configuration files and hyperparamters used for evaluation in the original paper was absent. This model being important since it was a primary tool used by the authors to show this behaviour of double descent in deeper models. We has to write this model and create a training script for this model specifically.

For deeper models, MCNN and ResNet, we could not observe the phenomenon of epoch-wise double descent without extreme label noise for Cifar-10 dataset. Given all the factors we were able to reproduce the results matching with the results of original authors' quite closely.

Future Work

Eliminating the double descent of pre-trained models or teacher student learning concept.
Since the over complex models such as neural networks show this behaviour; we may test the traditional machine learning models for example; clustering algorithms.
In some cases we would not need early stopping to achieve a model with less error, this can be a proper model selection problem to solve so we should check if the "elbow" method works for early stopping approach.

7 Work Division

Ashutosh Singh : I have done all of the sections relating to the MCNN model Two-Layer NN model and ResNet-18. For MCNN model I got the original training script working on multiple GPUs which is broken in the original repo along with multiple other errors. For two-layer NN I used the original authors notebook to be more user friendly. For ResNet-18 I created the training script for the model provided by Muhamet. This script is a modification of original authors' training code for MCNN. Any dependencies on $torch.nn.sequential$ are handled along with scaling of learning rates.

Burgunder Michal : I have done all of the sections relating to the Linear Least Squares model. I have taken the code that the original authors have included in their repo, and from the basic formulas that they have provided, extended a new script that analyzes the the formulas for a range of values. I have determined the ranges for each variable by manually determining where the limits of the risk function lie.

Muhamet Uslu : I have worked on the sections relating to the ResNet-18 model and gotten run MCNN model with the help of authors' and Ashutosh's repo and, I tested the given config files but it did not work for ResNet-18 structure then we decide to create our own model by using the PyTorch sequential class and finally I did literature review.

References

[1] Reinhard Heckel and Fatih Furkan Yilmaz. Early stopping in deep networks: Double descent and how to eliminate it. *CoRR*, abs/2007.10099, 2020.

[2] Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. Deep double descent: Where bigger models and more data hurt. *CoRR*, abs/1912.02292, 2019.

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

[4] L. Rosasco Y. Yao and A. Caponnetto. On early stopping in gradient, 2007.

[5] Garvesh Raskutti, Martin J. Wainwright, and Bin Yu. Early stopping and non-parametric regression: An optimal data-dependent stopping rule. *Journal of Machine Learning Research*, 15(11):335–366, 2014.

[6] Peter Bühlmann and Bin Yu. Boosting with the l2 loss. *Journal of the American Statistical Association*, 98(462):324–339, 2003.

[7] Yuting Wei, Fanny Yang, and Martin J. Wainwright. Early stopping for kernel boosting algorithms: A general analysis with localized complexities. *IEEE Trans. Inf. Theory*, 65(10):6685–6703, 2019.

[8] Madhu S. Advani and Andrew M. Saxe. High-dimensional dynamics of generalization error in neural networks. *CoRR*, abs/1710.03667, 2017.

[9] M. Soltanolkotabi M. Li and S. Oymak. Gradient descent with early stopping is provably robust to label noise for overparameterized neural networks.

[10] Wei Hu Zhiyuan Li Ruosong Wang Sanjeev Arora, Simon S. Du. Fine-grained analysis of optimization and generalization for overparameterized two-layer neural networks.

[11] R. Heckel and M. Soltanolkotabi. Denoising and regularization via exploiting the structural bias of convolutional generators.

[12] Reinhard Heckel and Fatih Furkan Yilmaz. Early stopping in deep networks: Double descent and how to eliminate it, 2020.

[13] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019.

[14] V. Nair A.Krizhevsky and G. Hinton. Cifar-10.