Analyzing the Double-descent Curve in Deep Learning

Miguel da Silva Almeida

***Abstract*—This paper aims to reproduce the double descent phenomenon described by Belkin [1], which suggests that neural networks will perform better when they have more trainable parameters than there are points in the training dataset used. We first introduce core concepts in understanding how a neural network is trained, before expanding into classical learning theory. This will allow us to review the classical approach to the bias-variance trade-off curve and how we reached this property. Then, we look at regularization, which can be a culprit in explaining the better performance for overparameterized regimes. After this theoretical introduction, we look at related work that shows that Belkin et al's [1] study is not an isolated occurrence, but one in many experiments that have originated discussion in the scientific community regarding the bias-variance trade-off. Among these, are results that aim to understand the reasoning behind the decreasing generalization error in overparameterized networks. Next we will show the results we obtained as we tried to replicate the double descent phenomenon, which led us to conclude that the phenomenon obtained by Belkin [1] is a consequence of the methodology used when training networks, resulting from different methods of initialization for under and overparameterized neural networks, namely the weight reuse strategy employed by the author, which leads to severe overfitting near the thresholds defined.**

***Index Terms*—Double descent curve, neural networks, machine learning, overparameterization.**

I. Introduction

A. Motivation

In the last years, there have been significant breakthroughs in machine learning, and increasingly complex models have materialized to allow studying richer data. On the core of developing these models, the concept of bias-variance trade-off [2] has been widely regarded as vital when it comes to deciding the simplicity of a model. It has been intrinsically related with underfitting and overfitting, meaning that models need to be rich enough to properly consider data in its entirety, but not too complex that they become too sensitive to noise while being trained. However, recent studies [3], [4] break this trend, by showing that neural networks perform better when they are trained to overfit the data. In this paper, we will study this theory further by analyzing the models used and studying whether neural networks break this classical theory of machine learning, or whether this comes from other underlying factors not considered when making these studies.

B. Goals

With the results presented in the paper by Belkin et al [1], classical approaches to the bias-variance trade-off have been outdone by a more accurate method. By taking classically overfitted models and expanding them further until better accuracy is achieved, the authors show there is a next step in the classic U-shaped bias-variance trade-off curve when the model capacity is increased beyond a certain point.

The goal of this project is to recreate this experiment, study the results obtained and, if possible to recreate, try to understand the theoretical reasoning behind it, whether there are some hidden features that make it fit the classical approach or there are further explanations to justify this phenomenon, shaking the core of the mathematical foundations for machine learning. The model will be studied before and after training, comparing its complexity in both stages. When studying them, we will also analyze if some hidden regularization happened that might justify the results obtained. If the experiment is impossible to recreate, try to understand whether there is some randomness associated with the experiment or there are some overlooks by the authors that make it invalid.

II. Theoretical Background

A. Neural Networks

A neural network is a model that processes information given to obtain a desired result. The input is fed to the different neurons in the network, which use mathematical functions to translate it into the final output. According to Bishop [5], a neural network can be simply described as "a series of functional transformations.".

A neural network is divided in different layers, each with its own neurons. These layers interact with each other by feeding information from one layer to another. When a lower layer feeds an output as the input of the next one, this is called *forward propagation*.

For each layer in the network, we define two initial parameters, $W^{[i]}$ and $b^{[i]}$, representing the *weights matrix* and *biases matrix*, respectively, where i represents the layer for which these parameters are initialized, with $i \in 1, ..., L$, where L represents the number of layers in our network. Using these parameters, we can define an initial transformation over our output as an equation:

$$Z^{[i]} = X^{[i]} \cdot W^{[i]} + b^{[i]}, \tag{1}$$

where $X^{[i]}$ represents a matrix which defines our input, and $Z^{[i]}$ represents the activation matrix.

This activation matrix will then be transformed using a function $h^{[i]}(x)$. These functions which are the core of a neural network are called *activation functions*. Through different activation functions, multiple behaviours can be simulated, and complex networks developed.

When using activation functions, we can distinguish them between linear and nonlinear functions. Linear functions are not widely used in deep learning for one main reason: all layers of the model can be collapsed into one, since each layer will be a linear function of the previous one, which means the final output will be a simple linear transformation of our initial input. With this problem, a model using linear functions has very limited flexibility and complexity.

The problems faced using linear functions can be avoided by using differentiable, nonlinear functions. These give a much greater flexibility to represent data through more complex functions, allow the use of backpropagation to train models, and allow a multiple layer architecture to be used to be able to process more complex data. The rectified linear activation, or ReLU for short, is a piecewise linear function that is widely used in deep learning. This function has a simple step function as its derivative, which makes computation much faster when doing backpropagation. This and the fact that, empirically, it has nearly identically accurate results to other more complex functions, makes this function very efficient. Other well known examples of nonlinear functions used are the sigmoid, the hyperbolic tangent, or tanH, and the softmax functions. Knowing this, we can define this transformation in the form

$$Y^{[i]} = h^{[i]}(Z^{[i]}). \tag{2}$$

This matrix *Y* effectively represents the output of layer *i* and is called a *hidden layer*, and will be fed as the input of layer *i+1*. A matrix $Y^{[i]}$ is the equivalent of the matrix $X^{[i+1]}$ as defined before.

This process of *forward propagation* represented in equations (1) and (2) is the method by which a feed-forward neural network obtains the predicted label for a certain input. Once layer *L* has been reached, the corresponding $Y^{[L]}$ matrix will represent our model's prediction.

B. Training

Now that we know how our network produces a result from a given input, we must learn how to train it to better fit our data and produce a correct result. This is done by minimizing what is known as an *error function*.

An error function helps measure how accurate the current model is when confronted with a training set. By analysing how far from correct the model's prediction is, we can later adjust it to better fit the problem at hand. Commonly in neural networks, the maximum likelihood approach is used. In many cases, this corresponds to minimizing the sum-of-squares error function. Thus, the error function would be represented by

$$E(W) = \frac{1}{2}\sum_{i=1}^{N}(y_n - t)^2 \tag{3}$$

where *y* represents the model's final prediction and *t* the correct label for the given input. In other cases, such as classification, the approach used would be using the cross entropy loss function.

When training a model, we take into consideration the loss value (how inaccurate the model's predictions are) of the training set. While the loss is high, the model cannot label the data and needs adjustments by updating the weight parameters of the network. Once the loss is low enough, we can stop the training process.

This raises the question: How does a model update its weights using these functions? To do this, a combination of forward and backwards propagation is applied, using the calculated loss and initial weights.

With the forward propagation, an initial prediction is obtained from a model to the input it received. When working with a training set, the correct labels from the inputs fed to a model are previously known. When comparing the predicted label and the real one, if the labels do not match, an error function is used to train the model to better predict it the next time. By applying these labels to the derivative of the error function, this information will be able to propagate backwards throughout all the layers, using the derivative of the activation function of each layer. When the initial layer is reached, using the transformations mentioned before, the weights and bias matrices will be updated which will update the model for its next predictions.

To effectively train a model into obtaining more accurate results, our objective is to minimize the error function, thus we try to find a minimum for this function. To do this, we start by looking at the gradient of the error function with respect to y.

Once the gradient is calculated, this result must be propagated throughout the network. This step is called *gradient descent*. We do so by updating the weights and bias matrices with a chain of equations propagated throughout successive partial gradient calculations throughout the network.

Using this chain of equations, we are capable of traversing our network backwards, updating the weights and biases of each layer, thus training our model given correct labels.

The most used version of this algorithm is the Stochastic Gradient Descent (SGD), where instead of performing these operations over all data points in our database, random points are chosen from the training set in order to perform these calculations, making this algorithm much more efficient.

C. Dimensionality

After understanding how we will develop and train a model, now we look at the data for which we want to train it. When analyzing a dataset, one of the most important things to take into account is its *dimensionality*.

Dimensionality represents the number of features we have on a dataset. When dealing with high dimensionality, the complexity of a problem becomes many times greater. This is known as the *curse of dimensionality*, a term coined by Bellman [6]. This problem can be easily understandable when we consider only the three geometrical dimensions. Having D representing the dimension of a problem, when $D = 1$ we have only one independent variable which can easily be represented by a line over one axis. Likewise, when $D = 2$,

we will have the space represented by the area within two axis and for $D = 3$ this space will be encapsulated in the volume between three axis.

While this can make it much harder to deal with higher dimensional spaces, normally real data can be effectively confined by using different feature engineering techniques. For this reason, when dealing with these problems, it is important to try to reduce the effective dimensionality of the data as much as possible and try to reduce the problem to a lower, equally representative, dimension.

D. Bias-Variance Trade-off

When given a wide variety of hypotheses and confronted with the possibility to still further develop new ones, we must know how to decide which one best fits our goals.

Considering that neural networks work over real valued targets, our problem will consider functions that map certain input labels to a real valued label, given by $f : X \to \mathbb{R}$. When looking at a certain hypothesis h, we revisit the squared error to decide how well it will fit our sample.

$$E_{sample}(h) = \frac{1}{N}\sum_{i=1}^{N}(h(x_n) - f(x_n))^2 \tag{4}$$

With this equation, we have a way to measure how far from the correct labels our hypothesis is. If we consider the whole population, then we now have a formula which helps us measure how accurate our hypothesis is by expanding it to the whole population.

$$E_{population}(h) = E_x[E_{sample}(h)] \tag{5}$$

Finally, we can then calculate the expected population error for a learning model which follows hypothesis h on a given sample D as the average population error of h for all possible training sets:

$$\mathbb{E}_D[E_{population}(h_D)] \tag{6}$$

Through the use of different properties, as expanded in Wichert and Sa-Couto [7], we derive a final formula for the expected error of a certain hypothesis:

$$\mathbb{E}_x[\mathbb{E}_D[(h_D(x) - \mathbb{E}_D[h_D(x)])^2] + (\mathbb{E}_D[h_D(x)] - f(x))^2]. \tag{7}$$

This formula can be divided into two very important terms for our future model analysis. The *variance* (21) and the *squared bias* (22).

$$VAR(x) = \mathbb{E}_x[\mathbb{E}_D[(h_D(x) - \mathbb{E}_D[h_D(x)])^2] \tag{8}$$

$$Bias^2(x) = (\mathbb{E}_D[h_D(x)] - f(x))^2] \tag{9}$$

Analysing both terms separately, we can see that variance represents how sensitive to noise a model is, i.e., a low variance means the results are similar when using training data or when using real data to test a model. High variance means the model was very sensitive to noise when being trained, and so it adjusted itself to arbitrary patterns on the training data, so it performs well on the training data but not on validation sets.

Bias represents how skewed a model is towards a particular result. It can be explained as a tendency from the network given by biased data or a very low training set.

To know how to properly decide how to fit a model, we must first understand how to detect high bias and high variance. High bias occurs in a model when the accuracy for both training and test sets is low, i.e., when the model is underfitting. This means it has not yet been trained enough to properly represent the data under study. On the other hand, as said above, high variance means the model is overfitting, being too sensitive to noise from the training data. This can happen when we try to get the loss value too close to zero when training a model.

This presents the problem on how to get a model that performs well on both sets, properly representing our data, and having a low bias, but also detaches itself from the training data, working properly as a standalone model for our generalized data, meaning it does not overfit. In figure 1, we have a visual representation of this trade-off in what is widely known as the bias-variance trade-off curve. We can see that by achieving a balance between variance and bias, we achieve a sweet spot when it comes to our model's complexity.

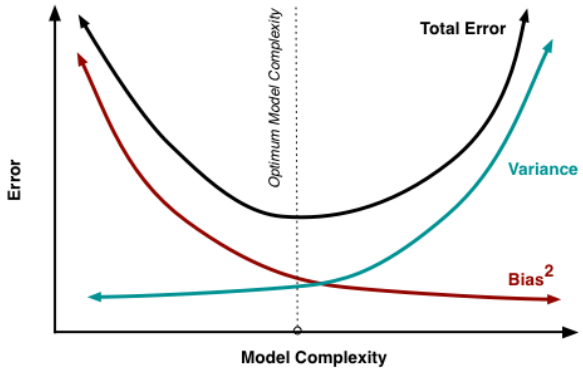


Fig. 1. Illustration of the trade-off between bias and variance. The optimum model will achieve perfect balance between both parameters. (Figure taken from [8])

E. Model Complexity

When talking about overfitting and underfitting, another key concept that arises is the concept of *model complexity*. When looking at a model that is too simple to properly represent our data, we say that this model is not complex enough, i.e., it underfits. Similarly, if it overfits our data, we see that the model is too complex.

To get a better understanding on how to effectively compute this complexity, we will look at one notion that tries to quantify it by measuring the model's capacity: the VC dimension [9].

When it comes to the VC dimension, we look at $\mathcal{H}$ as the hypotheses space, where $h \in \mathcal{H}$. This dimension represents the cardinality of the largest set that $\mathcal{H}$ can shatter.

A hypotheses space $\mathcal{H}$ is said to shatter a set of N points if for every possible combination of these points, there is a hypothesis $h \in \mathcal{H}$ that can correctly label them.

To measure the VC dimension, we must first find a set of k points that $\mathcal{H}$ can shatter and then find a set of $k+1$ points that $\mathcal{H}$ cannot shatter. What this will do is prove that $VC(\mathcal{H}) \geq k$ and $VC(\mathcal{H}) < k+1$, ultimately meaning that $VC(\mathcal{H}) = k$. Although the first part of this proof is generally easy, the second one is not. For this reason, we follow a simple heuristic that enables us to compute this dimension faster. This heuristic says that the VC dimension of a model is approximately the number of free learnable parameters of the model. In practice, this is what we will use to compute the complexity of our neural networks. Although the notion of VC dimension was developed with binary classification in mind, we will be applying the same logic to our study.

F. Regularization

As seen in the previous section, we ideally want our model to have a low enough bias such that it will be complex enough to correctly label our data, and a high enough variance so that it will be flexible enough to adjust to new data, but not high enough that it will overfit. This balance is easy to find in some cases, but the more complex our data is, the harder it becomes to find the perfect balance between bias and variance. For this reason, the concept of *regularization* was introduced. Regularization controls the rate at which our model trains itself and learns. By adding a coefficient to our learning function, it will discourage our model to learn a more complex or flexible model, tending to stay how it currently is. This helps dealing with noise in our training set and avoid overfitting. This is done by adding a term to our learning algorithm that will reduce the generalization error while having minimal impact in the training error, i.e., it will reduce the variance of our model with neglectable variance of the bias, thus resulting in an equally representative, but slightly less complex model. This type of trade-off results from regularization strategies based on regularizing estimators.

III. State of the art

Until now, we followed the classical machine learning approach, assuming the bias-variance curve as a fundamental guide to model selection [10]. This intuition is still largely followed by the scientific community and taught as a property from supervised learning. However, recent studies have started to refute this idea, particularly when dealing with neural networks, and have started questioning whether this notion applies at all.

In Neal [11], the author goes into detail about the intuition behind the bias-variance trade-off. It explains that by having a larger hypotheses space, we expect the variance to increase and the bias to decrease. This evolution stems from the fact that with a wider array of hypotheses to choose from, the likelihood that some approach an ideal function f to our problem increases. When looking at bias and variance as two separate terms, the author quotes Geman et al's [2] work, from which the decomposition arises.

$$\mathbb{E}_S R(h_s) = \mathcal{E}_{bias}(h_s) + \mathcal{E}_{variance}(h_s) + \mathcal{E}_{noise}. \tag{10}$$

From this formulation, we can see the direct, seemingly proportional, anticorrelation between bias and variance. If the average error remains constant and the bias varies, we expect the variance to vary as well to compensate this change, and vice versa. The author goes on to present the findings in which this trade-off stands and the reason why it is so commonly used. However, in Chapter 4, the refutation arguments begin. One of the main focus is the contrasting results when dealing with neural networks. Here, the author claims that some claims such as "bias falls and variance increases with the number of hidden units" is empirically false for many datasets. They go on to say that in some experiments made by Geman et al [2], the conclusions are misleading and too simplistic in determining the veracity of their claims.

To further defend their claim, Neal [11] skips ahead 20 years, to the more recent experiments made by Neyshabur et al [12] (which we will further develop later in this section), which prove that the opposite of Geman et al's [2] claims actually happen. Wider neural networks lead to a decrease in test error, which shows that when dealing with wider neural networks, the expected increase in variance in exchange for low bias does not necessarily happen.

Finally, the author takes Belkin et al [1] to show that there might indeed be a next step in the evolution of a neural network training. If we increase the hypotheses space enough, it will eventually lead to a decreasing test risk, while maintaining a training risk very close to zero. Although this experiment does not make the proper distinction between bias and variance, this decrease in test risk may further refute the claim made by Geman et al [2].

In Neyshabur et al [12], the authors try to explain the role of inductive bias as a capacity control parameter for neural networks, arguing that increasing the number of hidden units is not consistent with an increase in capacity. With this notion of inductive bias, the authors also state that implicit norm regularization also play a very important role in deep learning.

By studying a single-layer network, where the hidden layer has *H* rectified linear units, the authors state that by increasing the number of units *H*, normally we would reach what is known as the sweet spot in the bias-variance curve, where further increase of *H* leads to overfitting and thus an increase in test error. However, as stated before, and in accordance with the theory of the double-descent curve, what the authors observed is that by increasing *H*, they reached zero training error, but the test error continued decreasing, as seen in figure fig. 2. This experiment was done using stochastic gradient descent and no explicit regularization, which led the authors to believe that there was some implicit regularization happening. They reached this conclusion when, by manipulating the data

in the dataset to try and force overfitting, the network kept producing good results.

To try and explain this phenomenon, Neyshabur et al [12] consider a simple network with linear activation functions and only one hidden layer. When considering a network such as this one, controlling the size H corresponds to controlling the rank of the weights matrix W, as this model can be expressed as a matrix-factorization model, where $y = Wx = VU^T$. To regularize this model, we must only regularize the norm of V and U. Thus, the authors conclude that this norm will represent a better inductive bias than the number of weights. This leads them to conclude that, in reality, a network with an infinite number of hidden units would result in ideal performance.

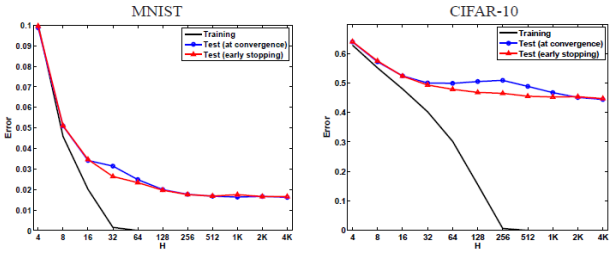


Fig. 2. Results obtained by Neyshabur et al [12] when analyzing the MNIST and CIFAR-10 databases. (Figure taken from [12])

In Geiger [13], the authors propose a theory for what happens in the transition between an under-parameterized model and an over-parameterized one.

Throughout their paper, they draw an analogy [14], [15] between deep neural networks and a random dense packing of repulsive particles. From this analogy they equate a neural network's units to particles in these systems, who only interact with each other within a certain range. In these systems, the energy is zero while there is still space for the particles to exist without interacting with each other and as soon as the system is full of particles, the energy will increase with density as new particles are added. For this comparison, the authors consider deep neural networks using the hinge loss. While the neural network is trained on a low enough number of points, the network can reach zero training loss. However, when increasing the training set size, the network becomes unable to achieve this perfect training. This transition happens when the number of points in our dataset becomes closer to the number of parameters in the network, hence the comparison with the aforementioned system.

In their paper [13], the authors reproduce the double-descent curve for the MNIST dataset using fully connected neural networks, which leads them to conclude that the border between under and overparameterization, which they call the "jamming point", is a crucial boundary when it comes to generalization error. What this means is that in under parameterized regimes, the generalization error slowly increases, meaning that models trained in these conditions will perform worse on real sets the closer they are to the jamming point. On the other hand, this behavior reverses in the overparameterized scenario, where the further away we go from the jamming point, the more this generalization error decreases. Regularization or early stopping in training these models make the double descent phenomenon disappear, which proves that the peak of curve relates to a very strong overfitting of the models.

In their study, the authors considered a plane of (N, P), where N represents the degree of freedom of a neural network, which through a comparison with the estimation of the VC dimension, can be considered the network's capacity, and P represents the size of the training set. In this plane, they established the jamming point $N^*(P)$, which depends on P but does not necessarily take the same value. So taking the authors' generalization assumption and an established (N, P) plane, intuition tells us that ideal network performance would be achieved as $N \to \infty$. If we consider their assumption, a network will constantly keep learning and the generalization error will still decrease as the capacity of the model increases, which pushes us towards this conclusion.

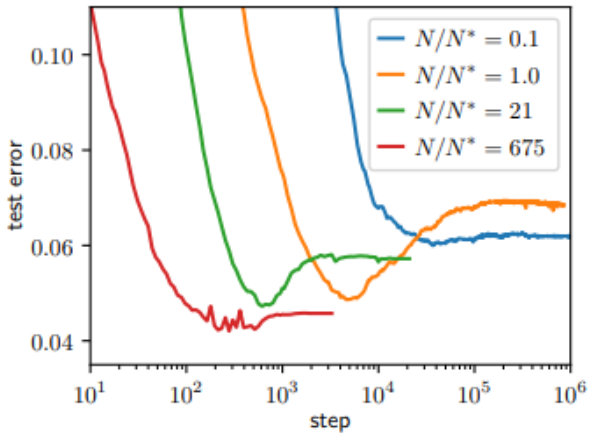
In figure fig. 3, the authors show the existence of this double descent curve on the MNIST dataset obtained under their training conditions. They show that for small and big enough datasets, overfitting has a weak effect on the test error without causing a significant increase on it, while for neural networks whose size was closer to the jamming point mentioned before, specifically when $N/N^* = 1$, the effect of overfitting causes a much more significant increase in test error.

With this result, the authors show that traditional methods of stopping training once a representative enough model has been achieved and not further increasing the model size is only a good practice until the jamming point, and that beyond this threshold models will tend to increasingly get better generalization error, thus achieving better results than the contrasting underparameterized models.

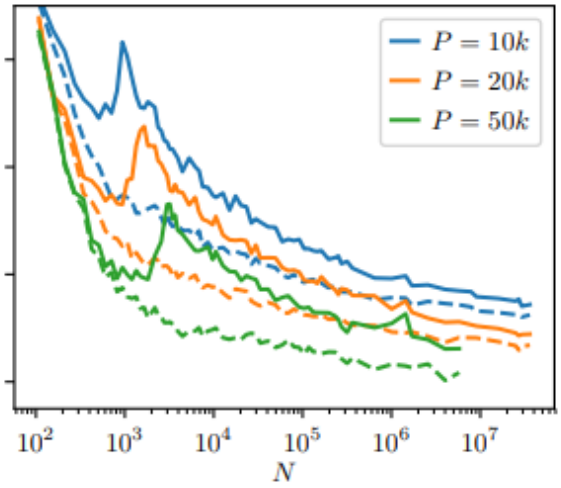
The authors conclude that this result is a consequence of constraint satisfaction when looking at the overparameterized regimes. They theorize that when a function's capacity goes over the number of points in the dataset, there are no constraints that remain unsatisfied, and thus the network cannot be stuck in a local minima, instead always being able to obtain the global minima. This means that for overparameterized regimes, a network that is trained enough over the training set will be able to find the best possible solution.

This result is backed by other recent studies [16]–[24], where many authors claim that overparameterized SGD leads towards a global minima when training neural networks.

In Advani [16], the authors show that for shallow networks initialized with small weights, larger networks generalize better than smaller ones for almost noise-free datasets. When looking at their results in figure fig. 4(b), when training a model with a low noise dataset, overparameterized models show no signs of overfitting, providing very good generalization results. However, if we look at figure fig. 4(a), for noisier settings, even though overparameterized models are able to fully learn the training dataset, they don't generalize as well as smaller models, especially those trained with early stopping. Additionally, the closer the size of the models gets to the overparameterized regime, the more important early stopping



(a) Evolution of different sized neural networks when trained over a period of time. Training was stopped when zero loss was achieved on training set.



(b) Test error at the end of training (solid line) and minimum error achieved (dashed line) as network size increases over different sized training sets.

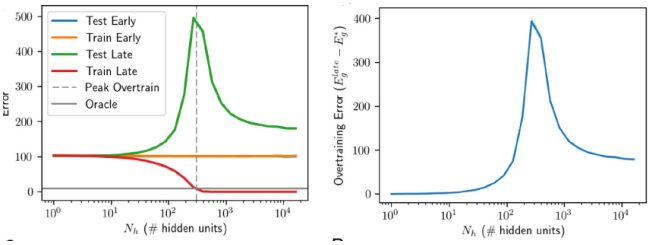
Fig. 3. Results obtained by Geiger [13] when analyzing the MNIST dataset with a 5 hidden layer fully connected neural network. (Figure taken from [13])

becomes for underparameterized regimes, since longer training leads to the most severe overfitting in this point. Even though it yields better results, this early stopping denies the double descent phenomenon as it prevents the near-transition point overfitting.

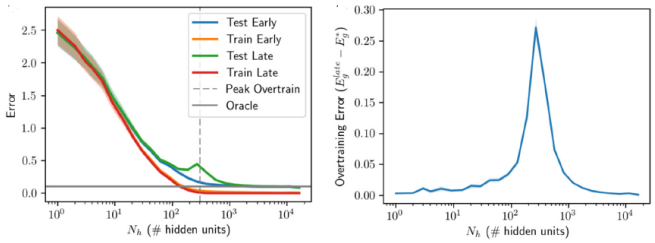
With these results, the authors show that for shallow networks, overparameterized networks operate ideally in low-noise settings, where larger networks will generalize significantly better than underparameterized networks.

IV. Results

In Belkin et al's [1] work, the authors propose that an overparametrized network yields much better results than an



(a) Results on random labelled data.



(b) Results on almost noise-free labels.

Fig. 4. Results obtained on a randomly generated dataset by Advani [16].

underparametrized one, thus surpassing the common practice of achieving the "sweet spot" in the classically considered bias-variance curve [25], [26] and choosing this model as the best.

During my work, my main goal was to reproduce the double descent curve proposed by Belkin et al and discover the underlying mechanics that exhibited during this phenomenon.

To replicate this phenomenon, I focused my efforts on the MNIST [27] dataset. As a heavily studied dataset, this is a well-documented and tested dataset, which requires almost no data preparation that can lead to practical differences when comparing my models to the ones obtained in Belkin et al.

We developed our neural networks on a subset of this data, working with all 10 classes of the original 10 dataset, but only 4200 points for training and 4000 points for testing.

A. Naive approach

The first attempt to replicate the double descent curve is a simple 2-layer fully connected neural network. With this architecture, we have a hidden layer with a ReLU activation function and an output layer that simply performs a linear transformation, thus has no activation function.

The networks were trained to minimize the mean squared error using stochastic gradient descent with momentum defined to 0.95, an initial learning rate of 0.01 and a step decay of 10% every 500 epochs. To speed up training, batches of 256 points were used. Additionally, no explicit regularization was added. These models were all trained over 6000 epochs.

For each layer, Glorot initialization [28], which is an initializer that draws a uniform distribution based on the number of parameters of the layer, was used to set the initial value of the weights.

If we look at the results obtained in figure 5, we see we get a constant decrease in the validation loss while the trained

models approach 0 accuracy loss and then further models seem to stabilize in how low they can get this validation loss. The dashed line represents the interpolation threshold as defined by Belkin [1]. With this line, we can see that neural networks do tend to perform better after the overparameterization threshold has been achieved, but we do not see the double descent phenomenon manifesting itself here.

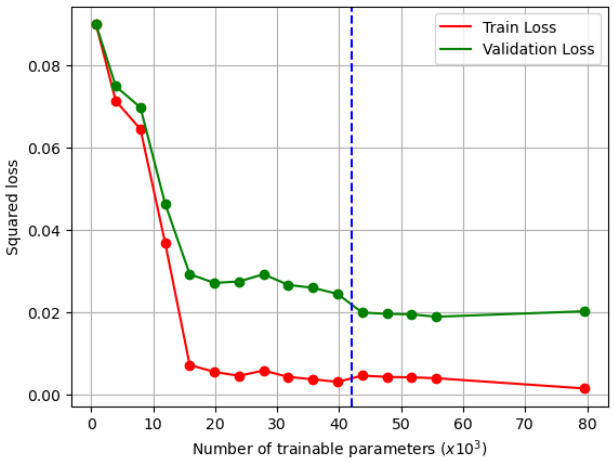


Fig. 5. Results obtained on the MNIST dataset following the naive approach described describing the squared loss value's evolution over the increasing number of parameters .

Other tests using increased depth and different loss functions were also performed and the double descent curve still did not manifest itself. Following this naive approach, it was not possible replicate the double descent curve as the models seemed to always improve with the added complexity.

B. Belkin's approach

In a second attempt to replicate the double descent curve, we employ the strategies employed in Belkin et al [1] while maintaining the architecture described in section 4.2.

As presented by the authors, different learning strategies were implemented before and after a threshold defined beforehand, which the authors call the "interpolation threshold". This threshold delimits the parameters of the neural network. To calculate the number of parameters in my network, I used the formula

$$\#Parameters = (I.H) + H + (H.C) + C \qquad (11)$$

where *I* represents the size of the input (in this case a 28x28 matrix of pixels), *H* represents the number of hidden units in the ReLU layer and *C* represents the number of classes, which coincides with the number of units in the output layer.

For neural networks smaller than the interpolation threshold, a strategy called "weight reuse" was used. An initial set of weights was defined using a Glorot initialization for the first network trained. From then on, all subsequent networks will be initialized with the learned weights of the previous network, where the added weights are sampled using normally distributed random numbers with a mean of 0 and variance of 0.01. The objective of this weight reuse strategy is to ensure the networks get as close as possible to the absolute minimum of the loss function they are trying to minimize. When using SGD, each step ideally gets our network closer to the intended minimum. However, with big enough steps this minimum can be overshot, while small enough steps might not learn enough relevant information to take us closer to the minimum. This strategy aims to tackle this problem by continuosly trying to get as close as possible to this minimum with subsequent networks. In all networks smaller than the interpolation threshold, a step size decay of 10% every 500 epochs has also been set to ensure it gets as close to a minimum as possible and an early stopping condition was set to stop training when 100% training accuracy was met.

For networks larger than the interpolation threshold, the weights were all initialized using the aforementioned Glorot initialization and a fixed step size throughout all training.

In both cases, training was stopped after 6000 epochs.

Under the conditions set in section 4.1, three different thresholds were set to test the consequences of using the weight reuse strategy defined by Belkin et al.

For the first step, a threshold on 16705 parameters was used. This corresponds to a neural network with a hidden layer with 21 units and an output layer with 10 units.

With the condition set, 16 models were trained before achieving the threshold and 5 models were trained after.

Under these conditions, Belkin et al [1] says that in the underparametrized regime we will witness the classic "U"-shaped curve, where we can observe overfitting, as the training loss decreases to values very close to 0, while the validation loss decreases until a certain point and then starts increasing. This well known behaviour occurs because the trained models are adjusting themselves too closely to the training data and developing a bias towards it. This will make them underperform in newly presented data, which translates in a higher validation loss.

If we look at the results obtained in figure 6, we can witness this behaviour in the graph to the left of the threshold line, where the models were trained using the weight reuse strategy. The solid line represents the treshold defined in training, while the dashed line represents the threshold as defined by Belkin [1].

After this expected behaviour, the authors propose that in the overparametrized regime, we will witness a sudden decrease in the validation loss while the training loss continues approaching 0. This also matches the results obtained, where we can observe this sudden descent to the right of the threshold line. In this step, the models were trained without weight reuse, with each model being trained independently.

For the second and third step of this experiment, we set the threshold to 31810 parameters and 39760 parameters, respectively. This corresponds to a neural network with 40 hidden units and 10 output units for the second step and a neural network with 50 hidden units and 10 output units for the third step.

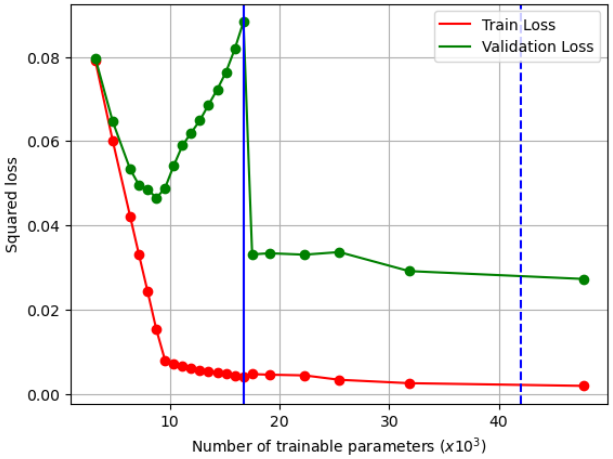


Fig. 6. Results obtained on the MNIST dataset using the weight reuse strategy with a threshold on 16705 parameters describing the squared loss value's evolution over the increasing number of parameters .

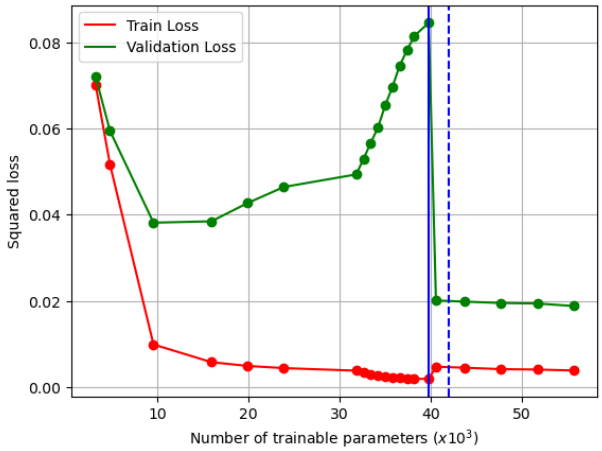


Fig. 8. Results obtained on the MNIST dataset using the weight reuse strategy with a threshold on 39760 parameters describing the squared loss value's evolution over the increasing number of parameters.

If we analyze the results obtained under these conditions in figures 7 and 8, we can easily identify the double descent curve phenomenon, where the peak of validation loss occurs always in the thresholds set, and after we witness the steep descent in validation loss, which approaches the training loss again.

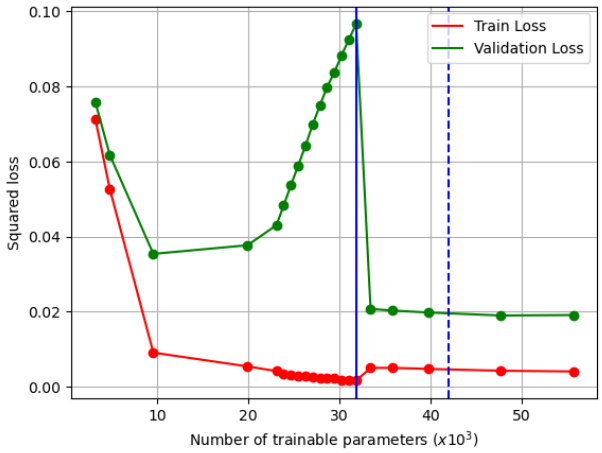


Fig. 7. Results obtained on the MNIST dataset using the weight reuse strategy with a threshold on 31810 parameters describing the squared loss value's evolution over the increasing number of parameters.

The authors propose that this double descent curve is intricately connected to the capacity of the function class (in our case the capacity of the neural network), where increasing this capacity beyond a certain threshold will yield the results obtained here.

According to the authors, this threshold is a fixed value given by

$$threshold = n.K \tag{12}$$

where n is the number of subsamples and K is the number of classes for training in the given dataset.

According to this equation, the double descent curve would be visible when our neural networks would have more than 42000 trainable parameters with the training dataset we are using. This does not match the results obtained. As we have seen, this double descent curve depends on the threshold set in training and not in a capacity dependent predefined threshold, as all the thresholds we have set are still smaller than the threshold proposed by the authors.

For this reason, we need to compare what is the difference between our naive approach, where we could not obtain the double descent curve, and Belkin's approach, where we could very clearly see the double descent curve shape reveal itself, however not as described by Belkin [1], but simply as a consequence of the weight reuse threshold defined by us.

C. Contrasting naive approach against Belkin's approach

When comparing the two learning strategies described before, the most obvious difference is the weight reuse approach used by Belkin [1], where a more complex model starts its learning with the weights of a previously trained, simpler model.

The first thing to notice when looking at this strategy defined by the authors is the mean and variance of the normal distribution from which they sample new weights to increase the matrix dimension for a more complex model. With a mean of 0 and a variance of 0.01, this distribution will generate weights extremely close to 0, which will have a very minimal impact in the model training.

When we look at section 2.2, we can see the rule to update our weights using gradient descent. We can see that the learning of lower layers depends on the gradients of the higher layers. With small weights, namely weights smaller than 1, these gradients will get smaller as we move backwards through our network, which will impact the learning of lower layers negatively, making them learn much slower. What this

means is, using this strategy, the layers will be impacted a lot more forcefully by the higher weights derived from the previous model than the added random weights. Realistically, while adding the extra complexity of more hidden units, the weight reuse strategy is just propagating an already learned model and keep trying to adjust more and more closely to the training data with the added complexity. This is why we can so clearly see the classically "U"-shaped curve in the trained models before the threshold. If we look at the results presented in Chapter III, namely the works done by Geiger [13] and Advani [16], we can see the authors concluded that in the presence of early stopping or other forms of regularization, the double descent curve did not show itself, only showing itself when the model was trained until severely overfitting, which is the behaviour we are forcing when using this weight reuse strategy. When we start learning a fresh model after this threshold, as it is expected, it yields better results than training a model with weights that were already starting to overfit the data. This explains the steep sudden decrease in validation loss that we obtain after the threshold where we stop using the weight reuse strategy.

If we look at the same architectures and same conditions applied before, but without the weight reuse strategy, we can see that the models before the defined threshold don't overfit in the given conditions and thus create better results as the model complexity increases, with this improvement steadily decreasing as we reach the overparametrized regime, after which simply increasing the capacity does not yield significant performance improvements.

This shows that for fully connected neural networks, the cause of this double descent curve phenomenon is not directly correlated with a sudden spike in performance after the threshold is achieved, but with a forced overfitting of the models before the given threshold, which increases the training risk to a peak before the weight reuse strategy ceases to be implemented.

If we look at what happens to learning when the weight reuse strategy is implemented the whole time, we see that our intuition is correct, as we can see that even past the threshold calculated before, the model's performance keeps slowly learning the training set. However, it keeps overfitting as the validation loss keeps increasing as we can see in figure 9.

Although these results do not prove or disprove the existence of the double descent curve, they highlight a clear flaw in the methodology used in Belkin's [1] work, which shows that at least under these conditions, the existence of this phenomenon is inconclusive.

V. Conclusions

In this document, we introduced the topic of the thesis which was developed throughout the course of this year, while explaining why these results might be extremely relevant to the current paradigm of deep learning, in particular when working with neural networks. We shed some insight over some basic concepts to properly analyze a neural network, as well as how to train it; we understood the intuition behind the bias-variance trade-off curve and understood why it could be extremely useful for model selection when working with models that follow this property and we saw the role regularization plays in regulating the capacity of our hypotheses space.

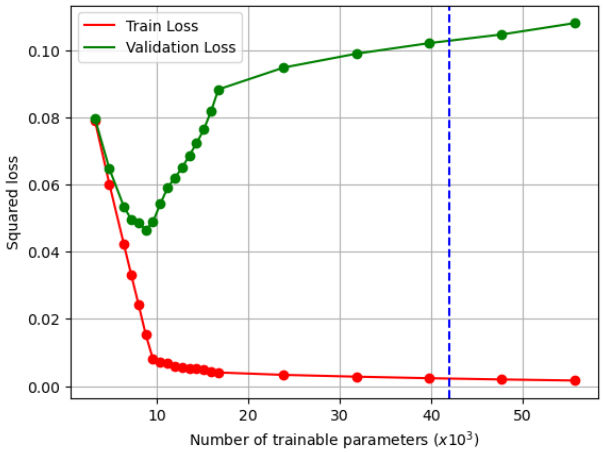


Fig. 9. Results obtained on the MNIST dataset using the weight reuse strategy throughout all models describing the squared loss value's evolution over the increasing number of parameters.

We also explored some related work that has been refuting the classical approach for some years, which worked as additional pillars to our research. In this work [13], [16], we have seen how generalization is affected in overparameterized neural networks.

Lastly, we presented our results when replicating the double descent curve as presented in Belkin [1]. We concluded that, while overparameterized networks do seem to achieve better generalization, the existence of the double descent curve in fully connected neural networks as presented in this paper is a consequence of the methodology used by the authors, namely the weight reuse strategy employed in underparameterized networks.

Even though the phenomenon presented by Belkin [1] seems to be a consequence of the methods used, their conclusions over overparameterized networks and how they generalize better seem to hold some truth. For this reason, we suggest that further analysis need to be done in what could be causing this by studying overparametrized networks, and whether this slight improvement proves true *ad infinitum* or if there is a limit to the generalization benefits on overparametrized networks.

References

[1] M. Belkin, D. Hsu, S. Ma, and S. Mandal, "Reconciling modern machine-learning practice and the classical bias–variance trade-off," *Proceedings of the National Academy of Sciences*, vol. 116, no. 32, pp. 15 849–15 854, 2019.

[2] S. Geman, E. Bienenstock, and R. Doursat, "Neural networks and the bias/variance dilemma," *Neural computation*, vol. 4, no. 1, pp. 1–58, 1992.

[3] B. Neal, S. Mittal, A. Baratin, V. Tantia, M. Scicluna, S. Lacoste-Julien, and I. Mitliagkas, "A modern take on the bias-variance tradeoff in neural networks," *arXiv preprint arXiv:1810.08591*, 2018.

[4] P. Nakkiran, G. Kaplun, Y. Bansal, T. Yang, B. Barak, and I. Sutskever, "Deep double descent: Where bigger models and more data hurt," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2021, no. 12, p. 124003, 2021.

[5] C. M. Bishop and N. M. Nasrabadi, *Pattern recognition and machine learning*. Springer, 2006, vol. 4, no. 4.

[6] R. Bellman, "Adaptive control processes, 255 princeton university press," *Princeton, New Jersey*, 1961.

[7] A. M. Wichert and L. Sa-Couto, *Machine Learning-A Journey To Deep Learning: With Exercises And Answers*. World Scientific, 2021.

[8] S. Fortmann-Roe, "Understanding the bias-variance tradeoff," *URL: http://scott. fortmann-roe. com/docs/BiasVariance. html (hämtad 2019-03-27)*, 2012.

[9] V. Vapnik, E. Levin, and Y. Le Cun, "Measuring the vc-dimension of a learning machine," *Neural computation*, vol. 6, no. 5, pp. 851–876, 1994.

[10] L. Prechelt, "Early stopping-but when?" in *Neural Networks: Tricks of the trade*. Springer, 1998, pp. 55–69.

[11] B. Neal, "On the bias-variance tradeoff: Textbooks need an update," *arXiv preprint arXiv:1912.08286*, 2019.

[12] B. Neyshabur, R. Tomioka, and N. Srebro, "In search of the real inductive bias: On the role of implicit regularization in deep learning," *arXiv preprint arXiv:1412.6614*, 2014.

[13] S. Spigler, M. Geiger, S. d'Ascoli, L. Sagun, G. Biroli, and M. Wyart, "A jamming transition from under-to over-parametrization affects generalization in deep learning," *Journal of Physics A: Mathematical and Theoretical*, vol. 52, no. 47, p. 474001, 2019.

[14] M. Wyart, "On the rigidity of amorphous solids," *arXiv preprint cond-mat/0512155*, 2005.

[15] A. J. Liu, S. R. Nagel, W. Van Saarloos, and M. Wyart, "The jamming scenario-an introduction and outlook," *arXiv preprint arXiv:1006.2365*, 2010.

[16] M. S. Advani, A. M. Saxe, and H. Sompolinsky, "High-dimensional dynamics of generalization error in neural networks," *Neural Networks*, vol. 132, pp. 428–446, 2020.

[17] S. Oymak and M. Soltanolkotabi, "Toward moderate overparameterization: Global convergence guarantees for training shallow neural networks," *IEEE Journal on Selected Areas in Information Theory*, vol. 1, no. 1, pp. 84–105, 2020.

[18] M. Soltanolkotabi, A. Javanmard, and J. D. Lee, "Theoretical insights into the optimization landscape of over-parameterized shallow neural networks," *IEEE Transactions on Information Theory*, vol. 65, no. 2, pp. 742–769, 2018.

[19] L. Venturi, A. S. Bandeira, and J. Bruna, "Spurious valleys in two-layer neural network optimization landscapes," *arXiv preprint arXiv:1802.06384*, 2018.

[20] Y. Li and Y. Liang, "Learning overparameterized neural networks via stochastic gradient descent on structured data," *Advances in neural information processing systems*, vol. 31, 2018.

[21] Z. Allen-Zhu, Y. Li, and Y. Liang, "Learning and generalization in overparameterized neural networks, going beyond two layers," *Advances in neural information processing systems*, vol. 32, 2019.

[22] Z. Allen-Zhu, Y. Li, and Z. Song, "A convergence theory for deep learning via over-parameterization," in *International Conference on Machine Learning*. PMLR, 2019, pp. 242–252.

[23] S. S. Du, X. Zhai, B. Poczos, and A. Singh, "Gradient descent provably optimizes over-parameterized neural networks," *arXiv preprint arXiv:1810.02054*, 2018.

[24] S. Du, J. Lee, H. Li, L. Wang, and X. Zhai, "Gradient descent finds global minima of deep neural networks," in *International conference on machine learning*. PMLR, 2019, pp. 1675–1685.

[25] V. Vapnik, *The nature of statistical learning theory*. Springer science & business media, 1999.

[26] R. Kohavi, D. H. Wolpert *et al.*, "Bias plus variance decomposition for zero-one loss functions," in *ICML*, vol. 96, 1996, pp. 275–83.

[27] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[28] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, 2010, pp. 249–256.