Double Descent
The Success Recipe Behind Large Neural Networks

Leonidas Ntrekos s6210058
Email: s6210058@aueb.gr
GitHub: LNtrekos

January 2026

Abstract

Double Descent as introduced by [Belkin et al., 2019] is a phenomenon that has arisen in relatively recent years due to the scaling of large neural networks. It was found that continually adding parameters to a model—even to the point of overfitting—could bring test error to an acceptable level of accuracy. It is a phenomenon that generated a storm of publications and discussion because it contradicts one of the core foundations of Machine Learning: The bias-variance trade-off. It is a description of why large neural networks -and in extent large language models- work so effectively, but a complete theoretical explanation is yet to be found. The scope of this paper is to demonstrate the aforementioned subject of interest.

1 Introduction

In recent years, the machine learning and artificial intelligence community has shifted its focus to scaling large neural networks to a great extent. Rightfully so, as their effectiveness and high accuracy in dealing with real-world problems have been repeatedly confirmed. The largest of them are large language models, the various so-called AI chatbots. ChatGPT, DeepSeek, Claude-AI, etc., basically are large -very large, with billions of parameters- Neural Networks that seem to be working really well in practice. Their deployment took most of the academics by surprise, as they were not expecting such good results. One of the reasons for that is the bias-variance trade-off, which, in plain English, states that if we try to explain a phenomenon we observe (train dataset) too much in detail (by adding too many parameters in a model -overfitting-), the same explanation (essentially the model) fails to capture the new circumstances of the same phenomenon (test dataset).

Nonetheless, scientists and practitioners took the initiative, adding too many parameters, and achieved great results, contradicting previously held beliefs. This brought a storm of both publications and discussions among people who work in the related field, as this finding shook one of the core foundations of statistical and machine learning, which is the bias-variance trade-off. In this paper, the bias-variance trade-off will be explained intuitively, with examples of the phenomenon presented.

2 Bias-Variance Trade-Off

Generally in statistical and machine learning theory, and more specifically in the supervised setting the goal is to obtain a predictor $f : \mathbb{R}^d \to \mathbb{R}$ given a sample of training examples $(x_1, y_1), \ldots, (x_n, y_n)$ from $\mathbb{R}^d \times \mathbb{R}$ that is used to predict the value y (regression) or the label y both given a point new point x, unseen in training. For formal definitions and a thorough explanation, refer to [Lafon and Thomas, 2024].

The classic approach in machine learning revolved around the idea of finding the so-called "sweet spot", which is the point where the train and test error are simultaneously minimized. The point where both over-fitting and under-fitting are avoided, the spot where the model is constructed can generalize well without being over-simplified, as mentioned in [Belkin et al., 2019].

In statistical and machine terminology, the error that we referred to above is usually measured through *mean square error* or *MSE* for short, given by

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{f}(x_i)\right)^2, \tag{1}$$

where $\hat{f}(x_i)$ is the fit that the estimated model $\hat{f}$ gives in the point for the ith observation, the x_i. The y_i represents the true realization and the value we wish to predict of the ith observation, given its corresponding feature x_i. The intuition behind this amount is that we try to measure how well the estimated model is performing by computing how far it is from reality. The smaller the MSE, the closer the estimate is; the larger the MSE, the farther it is.

The square is used to account for different signs (positive or negative) and is preferred to absolute due to its good mathematical properties. Additionally, highlights significant differences while minimizing the slight differences (big differences get bigger, small differences get smaller).

That is, in the regression setting, if we are concerned with a classification problem, the equivalent metric is the proportion of misclassifications when applying the estimated $\hat{f}$ to the ith observation:

$$\frac{1}{n}\sum_{i=1}^{n} I(y_i \neq \hat{y}_i). \tag{2}$$

Here $\hat{y}$ is the fitted class label given by $\hat{f}$. And $I(y_i \neq \hat{y}_i)$ is an *indicator variable* that equals to 1 if $y_i = \hat{y}_i$ and 0 if $y_i \neq \hat{y}_i$. If $I(y_i \neq \hat{y}_i) = 0$ then the ith observation was classified correctly by our classification method; otherwise it was misclassified. Hence, equation 2 computes the fraction of incorrect classifications.

Usually, we are interested in obtaining an average of the above error quantities. That is why, across different datasets, the MSE would yield different outcome values. So it would be helpful to know the mean value of MSE we would get by repeating the estimation of f. It is proven that the expected test MSE, for a given value x_0, can always be decomposed into the sum of three fundamental quantities: the *variance* of $\hat{f}(x_0)$, the *squared bias* of $\hat{f}(x_0)$ and the *variance* of the error term ϵ. That is,

$$\mathbb{E}\big(y_0 - \hat{f}(x_0)\big)^2 = \text{Var}\big(\hat{f}(x_0)\big) + \big[\,\text{Bias}\big(\hat{f}(x_0)\big)\big]^2 + \text{Var}(\varepsilon). \tag{3}$$

The amount $\mathbb{E}\big(y_0 - \hat{f}(x_0)\big)^2$ corresponds to the average MSE described above. Now, consider a given estimate $\hat{f}$ and a set of predictors X, which yields the prediction $\hat{Y} = \hat{f}(X)$. Then an alternative way to write 3 is the following:

$$\mathbb{E}(Y - \hat{Y})^2 = \mathbb{E}[f(X) + \varepsilon - \hat{f}(X)]^2 = \underbrace{[\,f(X) - \hat{f}(X)\,]^2}_{\text{Reducible}} + \underbrace{\text{Var}(\varepsilon)}_{\text{Irreducible}} \tag{4}$$

Equations 3 and 4 explain that the performance of the model depends on a reducible and an irreducible quantity. The first decomposes into the variance and the bias of the estimated function (or model) $\hat{f}$, which to minimize, must **simultaneously** achieve *low variance* and *low bias*. Note that variance is inherently a nonnegative quantity, and squared bias is also nonnegative. Hence, we see that the expected test MSE can never lie below $Var(\epsilon)$, the irreducible error from 4.

What do we mean by the variance and bias of a statistical or a machine learning method, though? *Variance* refers to the range of results that a statistical or a machine learning method

$\hat{f}$ would provide when estimating it in different datasets. On the other hand, *bias* refers to the error that is introduced by approximating a real-life problem, which may be highly complicated, by a much simpler model. In general, as we use more flexible methods, the variance will increase and the bias will decrease. Note that the change between the two quantities is not canonical, so both the rate and the relationship between the two may vary from problem to problem. That is, in short, the bias-variance trade-off. The above were obtained almost unchanged from the book *Introduction to Statistical Learning* ([James et al., 2014]). Readers are referred to that source for a more thorough exposition.

3 Double Descent

In modern machine learning techniques, however, it has been observed multiple times (see [Lafon and Thomas, 2024] for more details) that large models can generalize well despite fitting the training data almost perfectly. Running estimating algorithms, eg, Stochastic Gradient Descent (SGD for more information refer to [Ma et al., 2018], until zero training error often gives good out-of-sample error. So does this mean that overfitting and the usual bias-variance trade-off do not apply to enormous modern machine learning methods?

A widely adopted rule of thumb in modern deep learning is that network architectures should be designed with enough capacity so that the model can fit the training data perfectly, reaching essentially zero training error (often referred to as interpolation point [Belkin et al., 2019]). Interestingly, this contradicts the traditional bias–variance trade-off viewpoint. A growing body of empirical work shows that both neural networks and kernel-based methods can achieve excellent generalization performance even when they fully memorize the training set, including cases where the training set contains substantial noise. All of the above are summarized and illustrated in the plot below,

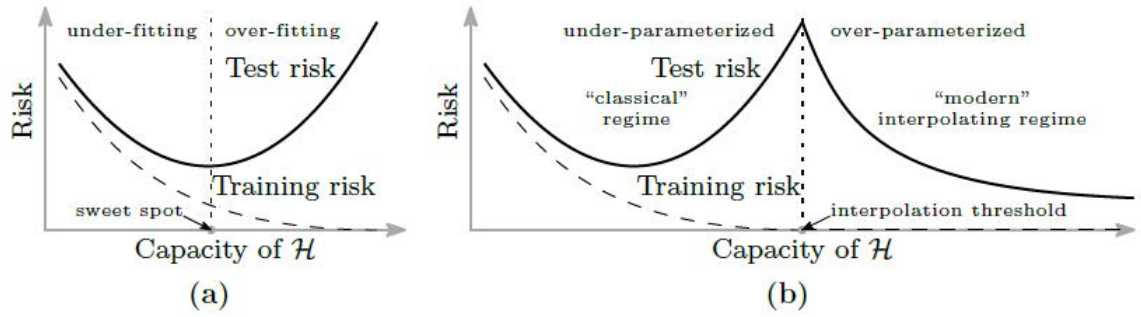


Figure 1: Plot taken form [Belkin et al., 2019], comparing the curves of **train set error** and **test set error**. **(a)** Corresponds to the classical U-shaped curve due the bias-variance trade-off. **(b)** Is an illustration of the double descent phenomenon, where the test error is lowered beyond the interpolation point.

Essentially, the plots describe the journey of training a large model. At first (a), both train and test errors are high, as the model's parameters increase, they start to decline until the "sweet spot" is reached, the point where simultaneously train and test errors are minimized. If the capacity of the function f (in the original paper [Belkin et al., 2019] where the plot was taken from the model was named $\mathcal{H}$ and the error was named Risk) keeps increasing, the test error "shoots through the roof" while the train error tends to zero. Note that the rates at which the two errors change differ. So far, we are in the classical statistical and machine learning regime, but if parameters keep being added to the model f beyond the interpolation point (the point where train error is zero), we enter (b) the modern setting where the double descent occurs.

That is, the test error comes back down to each prior values while the train error is kept at zero, as is demonstrated in 1.

In the classical setting (a), the balance of the fit is found at the bottom of the U shape. Everything to the left is considered under-fitting (high test error/risk), and everything to the right is considered over-fitting (again high test error/risk), while the train error is steadily decreasing as essentially being exactly memorized. The Double Descent, the phenomenon that has emerged in recent years, demonstrates that if parameters are added to the model beyond the interpolation point, where the model fits the training data perfectly, the test error returns to prior values.

The rest of this paper will demonstrate this phenomenon with empirical examples.

4 Cubic Natural Splines

In this section, a clear yet illustrative example from the book of [James et al., 2014] will be demonstrated to portray double descent through simulation. Let $x \sim U[-5, 5]$, we will generate 20 training points from that distribution, and then we will set $y = \sin(x) + \epsilon$ with $\epsilon \sim N(0, 0.3^2)$. Then we will fit a spline to the data with d degrees of freedom as:

$$f(x) = \hat{\beta}_1 N_1(x_i) + \hat{\beta}_2 N_2(x_i) + \cdots + \hat{\beta}_d N_d(x_i) = \sum_{j=1}^{d} \hat{\beta}_j N_j(x_i) \tag{5}$$

where N can be chosen to be any linear or nonlinear function. The main idea is that instead of fitting a single high-degree polynomial over the entire range available, we fit low-degree polynomials to different subsets (for more information [James et al., 2014] chapter 7.4).

For the following example, N's degree was chosen to be $d = 3$, that is, cubic polynomials. So the simulation is as follows: for 20 points generated from a uniform $U[5, 5]$ that we passed to $y = \sin(x)$ with some little added noise $\epsilon \sim N(0, 0.3^2)$, we fit the data exactly. In other words, we fit the model until the interpolation point, that is, until zero train data error, for illustration:

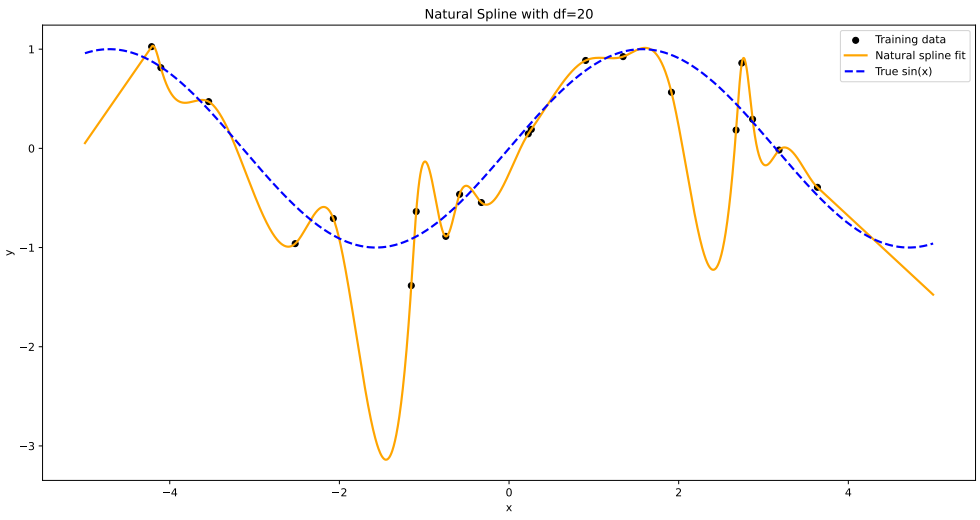
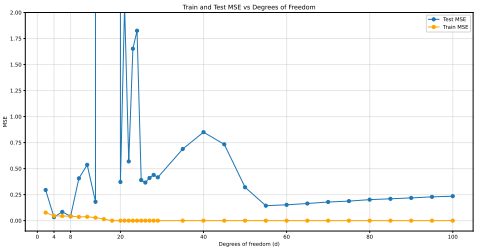


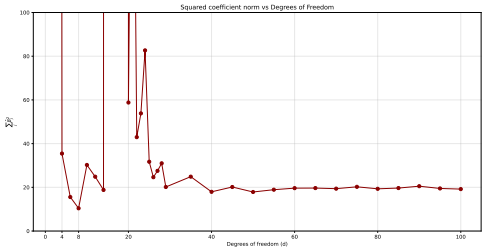
Figure 2: Perfectly Fitting 20 data training points with 20 different cubic polynomials.

The blue dashed curve shows the underlying $\sin(x)$ function that was used to create the data. The black dots indicate the 20 specific observations that served as the training set for fitting the model with some additional noise added from a Gaussian with $SD = 0.3$. Using only these 20

points to train the model, we then count the test error produced from 10000 points generated in the same way but without Gaussian noise. Essentially, we count the deviation we have from the original $\sin(x)$ curve. The results of this experiment are illustrated in the Figure below,



(a) Training and test MSE vs degrees of freedom



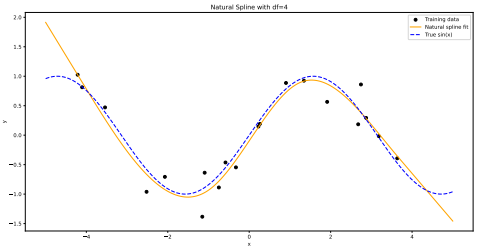
(b) ℓ_2 norm of spline coefficients

Figure 3: Model complexity diagnostics for natural spline regression

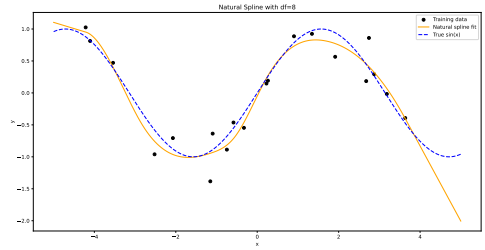
When the number of spline basis functions exceeds the sample size, the least squares problem becomes underdetermined. In this case, the fitted model corresponds to the minimum-norm interpolating solution, which is automatically selected by the Moore–Penrose pseudoinverse used in standard linear regression implementations.

The idea is that if you keep adding degrees, functions, parameters, even beyond the training points available, by finding the solution with the least norm, you will get a model that is able to generalize and predict new data satisfactorily.

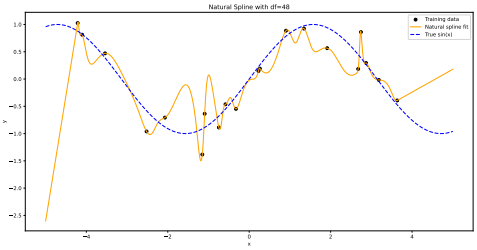
Below is an illustration of fitting different degrees to the same example, in which the degrees of freedom correspond to the number of cubic functions used to fit just the 20 training data, but used to predict 10000 unseen test data.



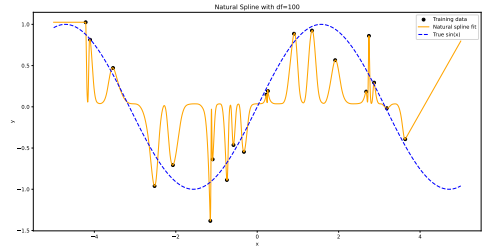
(a) df = 4



(b) df = 8



(c) df = 48



(d) df = 100

Figure 4: Natural spline fits for increasing degrees of freedom

Based on those plots, we can observe what is really happening. Recall figure 3, where we can see that both the train and test error are simultaneously minimized in 4 and 8 degrees of freedom. Those degrees, we can say, correspond to a sweet spot as referred to in [Belkin et al., 2019], as

they yield very good solutions as Figure 4 displays. At 20 degrees of freedom, see Figure 2, the solution really stretches itself and often goes to different places as it must fit all 20 training data points exactly. Then, at 42 degrees, it starts behaving better with smaller departures and smoother functions. Still making a few jumps in a few spots, but not as severe as when fitting exactly the 20 training data points. When we even increase to 100 cubic polynomials, we get these wiggle solutions that are not dramatically distant from the actual line when we wish to fit.

5 Random Fourier features.

We then consider another famous class of non-linear parametric models called *Random Fourier Features (RFF)*, which can be viewed as a class of two-layer neural network with fixed weights in the first layer as is discussed in [Belkin et al., 2019]. The RFF model family f with N (complex-valued) parameters consists of functions $f : \mathbb{R}^d \to \mathbb{C}$ of the form

$$f(x) = \sum_{k=1}^{N} a_k \phi(x; v_k) \quad \text{where} \quad \phi(x, v) := e^{\sqrt{-1}\langle v, x\rangle}$$

and the vectors $v_1, \ldots, v_N$ are sampled independently from the standard normal distribution in $\mathbb{R}^d$. Although the model is expressed in complex form, each complex coefficient can be decomposed into its real and imaginary parts. Hence, the function class can be regarded as a real-valued parametric family with $2N$ real parameters. Note that f is a randomized function class, but as $N \to \infty$, the function class becomes a closer and closer approximation to the Reproducing Kernel Hilbert Space (RKHS) corresponding to the Gaussian kernel.

Given a training sample $\{(x_i, y_i)\}_{i=1}^n$ with inputs $x_i \in \mathbb{R}^d$ and class labels $y_i \in \{1, \ldots, K\}$, we construct a Random Fourier Feature (RFF) representation associated with the Gaussian kernel. For a fixed number of random features N, each input x is mapped to a randomized feature space via

$$z(x) = \sqrt{\frac{2}{N}} \cos(Wx + b),$$

where the rows of $W \in \mathbb{R}^{N \times d}$ are sampled independently from $\mathcal{N}(0, \sigma^{-2} I_d)$ and the offset vector $b \in [0, 2\pi]^N$ is drawn uniformly at random. The class labels are encoded as centered one-hot vectors in order to remove the constant component. The predictor is obtained by minimizing the empirical squared loss over the RFF hypothesis class. In the over-parameterized regime, where multiple interpolating solutions exist, we select the minimum ℓ_2-norm solution, which is computed via least squares. For each value of N, we evaluate the training and test risks, as well as the ℓ_2 norm of the learned coefficients, while varying N across the under-parameterized, interpolating, and over-parameterized regimes. In the limit $N \to \infty$, this procedure converges to the minimum-norm interpolating solution in the Gaussian reproducing kernel Hilbert space, which serves as a reference predictor.

5.1 Random Fourier Features Approximation

The Gaussian kernel is shift-invariant and positive definite. By *Bochner's theorem* (see [Rahimi and Recht, 2007]), it admits the Fourier representation

$$k(x - x') = \int_{\mathbb{R}^d} e^{i\omega^\top (x - x')} \, p(\omega) \, d\omega,$$

where

$$p(\omega) = \mathcal{N}(0, \sigma^{-2} I_d).$$

This implies

$$k(x, x') = \mathbb{E}_{\omega \sim p(\omega)} \left[e^{i\omega^\top x} e^{-i\omega^\top x'} \right].$$

Using a Monte Carlo approximation, we draw

$$\omega_1, \ldots, \omega_D \sim \mathcal{N}(0, \sigma^{-2} I_d), \qquad b_1, \ldots, b_D \sim \text{Uniform}(0, 2\pi),$$

and define the random feature map

$$z(x) = \sqrt{\frac{2}{D}} \begin{bmatrix} \cos(\omega_1^\top x + b_1) \\ \vdots \\ \cos(\omega_D^\top x + b_D) \end{bmatrix} \in \mathbb{R}^D.$$

The kernel function is then approximated by the inner product

$$k(x, x') \;\approx\; z(x)^\top z(x').$$

Consequently, the nonlinear Gaussian kernel model can be approximated by a linear model in the random feature space,

$$f(x) = w^\top z(x),$$

where $w \in \mathbb{R}^D$ is learned by minimizing the empirical squared loss. In the over-parameterized regime, we select the minimum ℓ_2-norm solution, which can be computed efficiently via least squares. As the number of random features increases, this estimator converges to the minimum-norm interpolating solution in the reproducing kernel Hilbert space associated with the Gaussian kernel [Rahimi and Recht, 2008, Belkin et al., 2019].

5.2 MNIST example

We illustrate the behavior of Random Fourier Feature (RFF) models on the MNIST handwritten digit classification task. The dataset consists of grayscale images of size 28×28, which are vectorized into $d = 784$-dimensional feature vectors and rescaled to lie in $[0, 1]$. From the full dataset, we randomly select $n = 1000$ training samples and 1000 test samples. The task is treated as a $K = 10$-class classification problem, with class labels encoded as one-hot vectors.

Each input $x \in \mathbb{R}^d$ is mapped to a randomized feature representation $z(x) \in \mathbb{R}^N$ using Random Fourier Features associated with the Gaussian kernel. Specifically, the rows of the random weight matrix are sampled independently from $\mathcal{N}(0, \sigma^{-2} I_d)$, and random phase offsets are drawn uniformly from $[0, 2\pi]$. The resulting feature map is given by

$$z(x) = \sqrt{\frac{2}{N}} \cos(Wx + b).$$

Given the transformed training data, we learn a linear predictor in the random feature space by minimizing the empirical squared loss between the predicted scores and the one-hot encoded class labels. No explicit regularization is used. In the over-parameterized regime, where N exceeds the number of training samples, this optimization problem admits infinitely many interpolating solutions; among these, the least-squares solver selects the minimum ℓ_2-norm solution. Classification is performed by assigning each input to the class with the largest predicted score.

We vary the number of random features N across under-parameterized, interpolating, and over-parameterized regimes, and evaluate both training and test performance. In addition to the squared loss, we report the classification error (zero-one loss) and the ℓ_2 norm of the learned coefficient matrix. As N increases, the learned predictor approaches the minimum-norm interpolating solution in the reproducing kernel Hilbert space associated with the Gaussian kernel, in line with the theoretical framework studied in [Belkin et al., 2019].

Although the overall qualitative behavior of the curves is consistent with the interpolation-based picture described in the literature (see [Belkin et al., 2019] and Figure 1), the plots below

do not perfectly exhibit a smooth double descent phenomenon. This is primarily due to the relatively small sample size used in the experiment, with only $n = 1000$ training observations. If we were to match the exact experiment performed in [Belkin et al., 2019] with $n = 10000$ training examples, we would probably yield the same results.

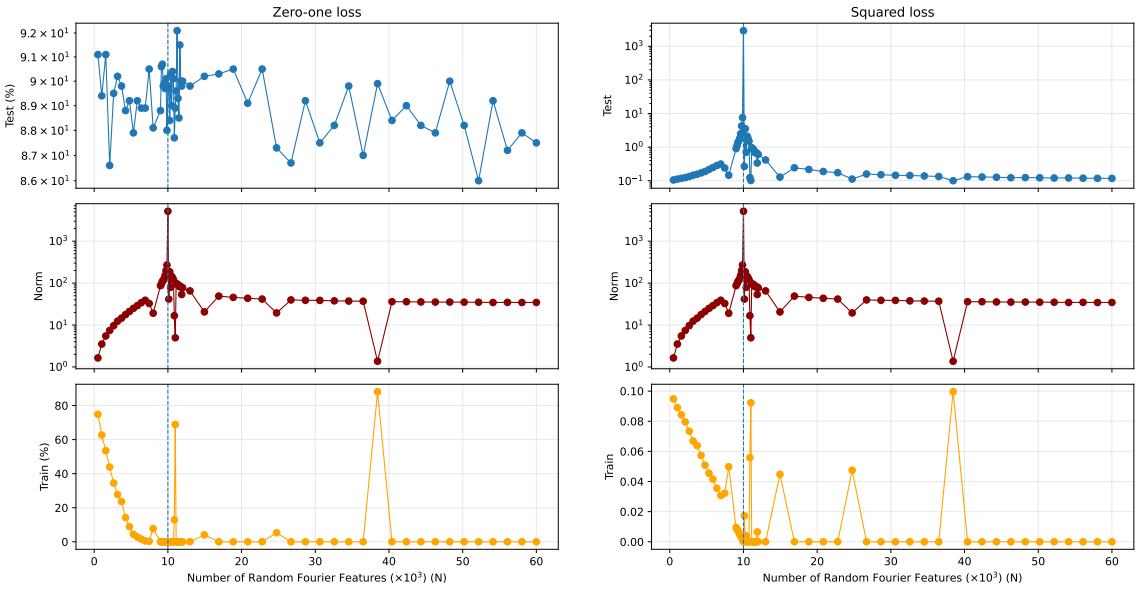


Figure 5: Training and test performance of Random Fourier Feature models on the MNIST dataset as a function of the number of random features N. We report classification error (zero-one loss), squared loss, and the ℓ_2 norm of the learned coefficients. The vertical dashed line indicates the interpolation threshold $N = n$, separating the under-parameterized and over-parameterized regimes.

6 Conclusion

We reviewed the classical bias–variance trade-off and discussed how recent empirical findings challenge its traditional interpretation in the context of highly over-parameterized models. Through a sequence of illustrative examples, including natural spline regression and Random Fourier Feature models, we demonstrated how predictors trained to interpolate the data can nevertheless exhibit good generalization performance.

In a wide Linear Model, namely in the case of predictors being much more than the observations ($p \gg n$), we fit a least squares solution using Stochastic Gradient Descent with a small step size, which leads to a minimum norm zero-residual solution. To train large neural networks, we use SGD to estimate their weights and get a more regularized solution. By analogy, deep and wide neural networks fit by SGD down to zero training error often give good solutions that generalize well.

Despite the various examples and analogies that are described, a full theoretical explanation remains to be found. So far, we have managed to explain why large machine learning models can generalize well, but not to grasp why exactly that is happening.

References

[Belkin et al., 2019] Belkin, M., Hsu, D., Ma, S., and Mandal, S. (2019). Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116(32):15849–15854.

[James et al., 2014] James, G., Witten, D., Hastie, T., and Tibshirani, R. (2014). *An Introduction to Statistical Learning: with Applications in R.* Springer Publishing Company, Incorporated.

[Lafon and Thomas, 2024] Lafon, M. and Thomas, A. (2024). Understanding the double descent phenomenon in deep learning.

[Ma et al., 2018] Ma, S., Bassily, R., and Belkin, M. (2018). The power of interpolation: Understanding the effectiveness of SGD in modern over-parametrized learning. In Dy, J. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 3325–3334. PMLR.

[Rahimi and Recht, 2007] Rahimi, A. and Recht, B. (2007). Random features for large-scale kernel machines. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 20.

[Rahimi and Recht, 2008] Rahimi, A. and Recht, B. (2008). Random features for large-scale kernel machines. *Advances in Neural Information Processing Systems*, 20:1177–1184.