Università di Genova
**DIBRIS** DIPARTIMENTO
DI INFORMATICA, BIOINGEGNERIA,
ROBOTICA E INGEGNERIA DEI SISTEMI

# Implicit Regularization For Deep Learning

Giacomo Garbarino

Master Thesis

Università di Genova, DIBRIS Via Opera Pia, 13 16145 Genova, Italy
https://www.dibris.unige.it/


**MSc Computer Science**
**Data Science and Engineering Curriculum**

# Implicit Regularization For Deep Learning

Giacomo Garbarino

Supervisor: Lorenzo Rosasco

Reviewer: Stefano Rovetta

December, 2023

# Table of Contents

# Abstract

Machine learning is one of the main methods employed to develop systems based on artificial intelligence. Supervised learning is a fundamental area of machine learning. In this context, machine learning models learn from labelled data through learning algorithms, along with loss functions and optimization algorithms, to make predictions on new data. Regularization and hyperparameter tuning are techniques to improve the model performance. Evaluation metrics are metrics to evaluate the performance of machine learning models. Statistical learning theory studies the behaviour of machine learning algorithms from a statistical point of view, describing how the error changes on test data according to different assumptions on the number of data points, on the data distribution and the models. Classical results in statistical learning theory describe the generalization error as composed of two parts: bias and variance. The bias decreases with more powerful models, but at the same time, the variance increases creating a tradeoff such that there exists an optimal intermediate model complexity which minimizes the generalization error. However, modern results demonstrate that in highly complex machine learning models the generalization error tends to decrease again when models interpolate the training data. We demonstrate that this phenomenon, called double descent, is not easily observable in modern neural networks and modern random feature models. We also demonstrate that these models tend to interpolate the training data and perform well on test data, they tend to be robust to high levels of noise and the choice of the loss function has no impact on the interpolation.

# Chapter 1

# Introduction

Artificial intelligence is the simulation of human intellect in computers, enabling them to carry out activities like learning that normally require human intelligence [KD20] [DDPR15]. Numerous industries, including healthcare, finance, education, transportation, and entertainment, are greatly impacted by artificial intelligence. It can improve decision-making, simplify procedures, resolve challenging issues, and automate monotonous work. Machine learning is one of the main methods used in the development of AI systems. The power of machine learning comes from its capacity to handle massive volumes of data, learn from it, and make predictions or decisions based on that learning. In the first chapter of the thesis, we explore the foundational ideas of machine learning. Supervised learning, a fundamental area of machine learning, is covered in the first section. In supervised learning, machine learning models, which comprise data and learning algorithms often with adjustable parameters and hyperparameters, gain insight from labelled data to generate predictions and make decisions. This thesis begins by describing the steps that a supervised learning model takes to learn from labelled data. These steps include training, validating, and testing which utilize a dataset split into a training set and a test set. The overview of loss functions and optimization algorithms, two key ideas in the machine learning model training process, will take up the next two parts. A specific kind of function called the loss function is employed to assess how well a supervised learning model performs throughout training. Conversely, optimization methods, which depend on the data from the loss functions, are used to adjust the parameters of a supervised learning system during training. A completely developed machine learning model, which represents the method, its learnt parameters, and the data used, is the result of this training process. This chapter concludes with a discussion of the evaluation metrics. These metrics are essential for evaluating and testing a trained and validated machine learning model performance. The data may show complicated, non-linear patterns or relationships in a lot of real-world situations. The related task is deemed "non-linear separable" when the data exhibits complex dependencies

or non-linear patterns. More complex strategies are needed to solve non-linear situations. This is where neural networks, random features, and kernel functions are useful. These methods are examined in the second chapter. Over the years, there have been significant improvements and changes to machine learning. The bias-variance tradeoff theory states that the generalization error of the machine learning model comprises two parts: the error from erroneous assumptions made by the algorithm (bias) and the error from sensitivity to small fluctuations in the training data (variance). This theory says that as the model complexity increases, the bias declines, while the variance increases, giving to the generalization error a U-shape pattern and a tradeoff between these two components of the generalization error is necessary to minimize it and ensure the model predicts well. In recent years, a new phenomenon has emerged; the appearance of a double descent curve [BHMM19]. This interesting phenomenon shows that the generalization error has a distinct pattern: as model complexity increases, it first declines, then it rises, and then it declines again, showing that the standard bias-variance tradeoff may not apply to certain problems and models. In the third chapter, we will introduce first regularization techniques and hyperparameter tuning; two important methods to enhance model performance. Then, we will examine the two theories and the research contributions of the famous professor of data science Mikhail Belkin. We will perform experiments based on some of his contributions in the upcoming chapter. These tests will realize the dual goals of demonstrating some of his ideas and confirming our hypotheses such that increasing the levels of noise in a model data means deteriorating further its generalization error, in the presence of high levels of noise the generalization error does not explode and does not get better if the model trained on noisy data is evaluated on noisy test data.

# Chapter 2

# Basics Of Machine Learning

*This chapter introduces the basic concepts of supervised learning; one of the most relevant machine learning subfields. It explains the concept of a supervised learning problem, what is a machine learning model and its general process of how to make him learn from data, with a particular focus on loss functions and optimization algorithms. In the end, it describes some metrics which allow us to evaluate the performance of the model.*

## 2.1 Supervised learning problem

Machine learning is a branch of artificial intelligence that focuses on building algorithms that let machines "learn" from data, without being explicitly programmed. Considering a supervised learning scenario, the dataset used by algorithms is a set of pairs $(x,y)$. Each pair is a data sample, or data point, where $x=(x_1,\ldots,x_n)$ is the set of input data, or features, while $y$ is the actual output value or simply actual output. Each sample has been generated according to a joint probability distribution $p(x,y)$. This probability is fixed, but unknown. Between the input and the output, there exists a target function such that $f(x)=y$. Since $p(x,y)$ is unknown, this function cannot be computed. The goal of a machine learning algorithm is to learn an estimator $f_{es}$ which best mimics the target function. The general process of learning is as follows. Consult Chapter 2 of this book [JWH+23] for additional information.

1. Data Preparation: Data is preprocessed to ensure it is in a format that can be used by an algorithm. This generally involves steps such as data cleaning and data normalization. Data used by machine learning algorithms are generally structured, such as data stored in a database or spreadsheet.

2. Feature Selection: Feature selection is the process of identifying and keeping only

the most relevant input data that will help the algorithm make accurate predictions. Irrelevant and redundant data may negatively impact and reduce the overall performance and accuracy of the algorithm.

3. Algorithm Selection: This step is to select the most appropriate machine learning algorithm based on the type of problem and the data available.

4. Data Split: Data is divided into two sets known as training set and test set. Each set has a specific purpose and a different size. Usually, the training set is seventy percent of the entire data.

5. Training: During the training phase, a machine learning model, which comprises the selected algorithm and the data, learns from the training set to make predictions. This process involves the following steps:

    5.1 Initialization: In the beginning, the internal parameters, or simply parameters, of the algorithm are initialized to some default or random values. Parameters are the internal variables of a machine learning algorithm that are learned during the training process. They are used to make predictions based on input data. These parameters depend on the type of machine learning algorithm being used. They can be weights, biases, maximum depths and many more.

    5.2 Feeding the Data: The algorithm is fed with the training samples, which consist of a set of features and their corresponding actual output values.

    5.3 Making Predictions: Based on the features and the initial parameter values, the algorithm makes predictions for the actual output values.

    5.4 Loss Function: Loss functions, also known as cost functions, are a class of functions to measure the difference between the predictions of the algorithm and the actual outputs. The goal of training is to minimize this difference or loss. The choice of loss function depends on the specific problem and the nature of the data.

    5.5 Updating the Parameters: The algorithm adjusts its parameters to minimize the loss between the predicted and true output values. This adjustment is done using an optimization algorithm, such as gradient descent, which finds the direction in which the loss decreases the most and updates the parameter values accordingly.

    5.6 Iterations and Batches: Steps 5.2 to 5.5 are repeated until the loss is minimized, or until a stopping criterion is met. In the training phase, the training dataset is divided into smaller batches. During each iteration through the entire dataset, the algorithm processes one batch at a time. The parameters of the algorithm are updated based on the predictions and the corresponding loss for that batch. This is more efficient than processing the entire dataset at once, especially when

dealing with large datasets, as it requires less memory and speeds up the training process.

6. Validation: This phase is used to evaluate the performance of the model. It uses a technique called cross-validation which splits the training set into multiple subsets for an auxiliary training and evaluation of the model performance. Most machine learning algorithms have external parameters called hyperparameters. These hyperparameters are not learned from the data but are set before the training process begins. They control various aspects of the training process and model architecture. Cross-validation utilizes the same loss function of the primary training phase to assess and adjust the hyperparameters of the model to find the best hyperparameter settings that optimize its performance. The hyperparameters that can be fine-tuned during this phase include the learning rate, batch size, the number of layers or units in a neural network, and more. The validation phase is an iterative process. The choice of stopping conditions depends on our goals, but this phase usually stops when the performance of the model starts deteriorating.

7. Testing: Once the model is trained and validated, it is evaluated on the test set to assess its generalization performance on new data. This assessment is done using various evaluation metrics. The choice of evaluation metrics depends on the specific problem and the nature of the data. For example, in classification tasks, metrics like accuracy, precision, recall, or F1 score may be used. In regression tasks, metrics like mean squared error (MSE), or mean absolute error (MAE) are commonly employed.

## 2.2 Loss functions

Loss functions, also known as cost functions, are a fundamental concept in machine learning. They are used to quantify the training loss; the discrepancy between the predicted values generated by a machine learning model and the actual output values in a training set. They can also be used to compute the test loss; the discrepancy between the predicted values generated by a machine learning model and the actual output values in a test set. The goal of a machine learning model is to minimize the output of this loss function, which, in turn, helps the model make more accurate predictions. The choice of a loss function depends on the specific type of machine learning task. Consult Chapter 7 of the book [JWH$^+$23] and the paper [CEL$^+$23] for additional information. In supervised learning, there exist two fundamental types of tasks:

- Classification Task (Chapter 4 of the book [JWH$^+$23]): Classification is a supervised learning task where the goal is to assign input data to predefined classes. These classes are often discrete. The output of a classification model is a value, label

value, or simply label, that indicates which class the input data belongs to. Typical classification tasks include classifying emails as spam or not spam, classifying animals, and classifying a tumour as benign or not benign.

- Regression Task (Chapter 3 of the book [JWH+23]): Regression is also a supervised learning task, but it deals with predicting continuous, numeric values. The output of a regression model is a value that indicates a quantity. Typical regression tasks include predicting house prices based on various features (e.g., size, location, number of bedrooms), forecasting stock prices, and estimating the age of a person based on biometric data.

In binary classification, labels are typically assigned the values 0 and 1 to represent the two mutually exclusive classes that the model is trying to predict. This convention is not a strict rule but a common practice to encourage simplicity and compatibility with existing tools. Here are some common types of loss functions for classification tasks. Consult Chapter 7 of the book [JWH+23] and the paper [CEL+23] for additional information.

- 0-1 Loss: 0-1 loss is a type of loss function used in the context of classification problems. It is a simple but often impractical loss function because it assigns a penalty of 1 for misclassifications and 0 for correct classifications. Given $N$ the number of data samples, $y_i$ the actual output for the i-th data sample, $\hat{y}_i$ the predicted value for the i-th data sample and $I$ the indicator function, which returns 1 if the condition inside is true and 0 otherwise, the 0-1 loss applied to the entire dataset is defined as:

$$L = \frac{1}{N} \sum_{i=1}^{N} I(y_i \neq \hat{y}_i) \tag{2.1}$$

  The 0-1 loss function is not practical in many real-world classification problems for several reasons. One of them is that the 0-1 loss function is non-differentiable. This means that its gradient concerning the model parameters cannot be computed. In contrast, the main optimization algorithms used in machine learning rely on gradients to update the parameters to minimize the loss function.

- Binary Cross-Entropy: Binary cross-entropy, often referred to as log loss, is a common loss function used in binary classification tasks in machine learning. It is used to measure the dissimilarity between predicted probabilities to belong to the actual label and the actual label. This loss function is particularly well-suited for binary classification problems. Given $N$ the number of data samples, $y_i$ the actual output for the i-th data sample and $\hat{y}_i$ the predicted value for the i-th data sample, the binary cross-entropy loss applied to the entire dataset is defined as:

$$L = -\frac{1}{N} \sum_{i=1}^{N} [y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)]. \tag{2.2}$$

In this formula, if $y = 1$, the loss is determined by $-log(\hat{y})$, which encourages the predicted probability $\hat{y}$ to be close to 1. If $y = 0$, the loss is determined by $-log(1-\hat{y})$, which encourages $\hat{y}$ to be close to 0.

- Sparse Categorical Cross-Entropy: Sparse categorical cross-entropy is a commonly used loss function in multi-class classification tasks. It measures the dissimilarity between the predicted probabilities to belong to the actual labels and the actual labels. Given $N$ the number of data samples, $y_i$ the actual output for the i-th data sample and $\hat{y}_i$ the predicted value for the i-th data sample, the sparse categorical cross-entropy loss applied to the entire dataset is as follows:

$$L = -\frac{1}{N}\sum_{i=1}^{N} y_i \log(\hat{y}_i). \tag{2.3}$$

Here are some common types of loss functions for regression tasks. Consult Chapter 7 of the book [JWH+23] and the paper [CEL+23] for additional information.

- Mean Squared Error (MSE): Mean squared error is a widely used loss function to measure the average squared difference between the actual outputs and the predicted values. It is a way to quantify how well the predictions match the actual output values. Given $N$ the number of data samples, $y_i$ the actual output for the i-th data sample, and $\hat{y}_i$ the predicted value for the i-th data sample, the MSE is defined as:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2. \tag{2.4}$$

  The result is a single non-negative number, with smaller values indicating a better fit between the predictions and the actual outputs.

- Root Mean Square Error (RMSE): Root mean square error is a regression loss function computed as the square root of the MSE. The formula for the RMSE is as follows:

$$RMSE = \sqrt{MSE}. \tag{2.5}$$

  RMSE is often preferred when you want a measure of the loss that is easier to relate to the magnitude of the data (the range of values of the data before computations), whereas MSE can be more useful in some mathematical and computational contexts where the squared values are advantageous (e.g. when computing gradients for optimization algorithms).

- Mean Absolute Error (MAE): Mean absolute error is another commonly used loss function to evaluate the performance of a regression model. It measures the average

absolute difference between the actual outputs and the predicted values in a dataset. MAE is less sensitive to outliers compared to Mean Squared Error (MSE). Given $N$ the number of data samples, $y_i$ the actual output for the i-th data sample, and $\hat{y}_i$ the predicted value for the i-th data sample, the formula for MAE is as follows:

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|. \tag{2.6}$$

The result is a single non-negative number, with smaller values indicating a better fit between the predictions and the actual outputs.

Applying a loss function to all data points returns the average loss of a model on the data. In particular, the 0-1 loss applied to the entire dataset is also a representation of the mean classification error. The mean classification error is a metric to evaluate model performance. Mathematically, they are both computed as the ratio of the number of wrong predictions to the total number of data points.

## 2.3 Optimization algorithms

Optimization algorithms play a crucial role in machine learning. They are used to adjust the parameters of machine learning models during the training to minimize a loss function, making the model better at making predictions. Here are some common optimization algorithms used in machine learning. Consult Chapter 10 of the book [JWH+23] and the paper [Rud17] for additional information.

- Gradient Descent (GD): Gradient descent is a fundamental optimization algorithm commonly used in machine learning to minimize the cost or loss function. The main idea behind gradient descent is to iteratively update the parameters of the model in the direction of the steepest decrease in the loss function, which helps the model to converge to a local minimum (or in some cases, a global minimum) of the loss function. Here is a step-by-step explanation of how gradient descent works in the context of machine learning:

  1. Initialization: In the beginning, the parameters of the model are initialized randomly or with some predefined values.
  2. Compute the Loss Function: The model calculates the value of a specific loss function $G$.
  3. Compute the Gradient: It computes the gradient of the loss function concerning each parameter. The gradient represents the direction and magnitude of the

steepest increase in the loss function. Given $\frac{\partial G}{\partial \theta}$ the partial derivative of the value of the loss function $G$ concerning the parameter $\theta$, the formula to compute the gradient of a loss function concerning a single parameter $\theta$ is:

$$\nabla G(\theta) = \frac{\partial G}{\partial \theta}. \tag{2.7}$$

4. Update the Parameters: The parameters of the model are adjusted in the direction opposite to the gradient. Given $\theta_{old}$ the current value of the parameter $\theta$, $\gamma$ the learning rate, and $\nabla G(\theta_{old})$ the gradient value of the loss function $G$ relative to the parameter $\theta$ to be updated, the update rule for a single parameter $\theta$ is given by:

$$\theta_{new} = \theta_{old} - \gamma \nabla G(\theta_{old}). \tag{2.8}$$

The learning rate is a parameter that controls the step size of parameter updates during the optimization process. It determines how large or small the steps are when adjusting the parameters of the model. In practice, it is calculated using cross-validation.

5. Repeat: Steps 2 to 4 are repeated for a fixed number of iterations or until a stopping criterion is met.

- Stochastic Gradient Descent (SGD): Stochastic gradient descent is a variant of the gradient descent optimization algorithm. Unlike traditional gradient descent, which computes the gradient of the loss function concerning the parameters using the entire training set in each iteration, SGD updates the parameters using only a single randomly chosen training sample at a time. Here is how stochastic gradient descent works:

1. Initialization: In the beginning, the parameters of the model are initialized randomly or with some predefined values.
2. Shuffle Data: It shuffles the training dataset to ensure that the order of the data samples does not bias the optimization process.
3. For each Training Sample Chosen Randomly:
   - 3.1 Compute the Gradient: It calculates the gradient of the loss function concerning the parameters for the current training sample. The formula to compute the gradient of a loss function $G$ concerning a single parameter $\theta$ remains the same (2.7).
   - 3.2 Update the Parameters: The parameters are adjusted in the direction opposite to the gradient. The update rule for a single parameter $\theta$ remains the same (2.8).
4. Repeat: Step 3 is repeated for all training samples for a fixed number of iterations or until a stopping criterion is met.

SGD has several advantages:

- Faster Updates: Because it processes one training example at a time, each iteration of SGD is typically faster than gradient descent.
- Escape Local Minima: Updates in SGD can be noisy. This can help the optimization process escape local minima in the loss function. The term "noisy updates" refers to the inherent randomness and variability in the gradient estimates and parameter updates that occur during each iteration. When the gradient of the loss function for a single training sample is computed, the estimated gradient can vary significantly from one sample to another, even if they are very similar. This variability is due to the randomness inherent in the selection of training samples.
- Generalization: It can lead to better generalization in some cases, especially when the dataset is large and noisy.

SGD has also some disadvantages:

- Noisy Updates: Noisy updates in SGD cause a lot of variance in the optimization process. As a result, it might take a longer time to converge.
- Unstable Convergence: SGD does not necessarily converge to a minimum of the loss function. It oscillates around the minimum region.
- Learning Rate Tuning: Proper tuning of the learning rate is critical, as an inappropriate learning rate can lead to slow convergence or divergence.

Variants of SGD, such as mini-batch gradient descent and Adam (short for "Adaptive Moment Estimation")[KB14], combine several features to improve the optimization efficiency and the speed of convergence.

## 2.4 Evaluation metrics

Machine learning evaluation metrics are used to assess the performance of a machine learning model. The choice of evaluation metrics depends on the specific problem and the nature of the data. Here are some common machine learning evaluation metrics. Consult the paper [HM15] for additional information.

- Accuracy: Accuracy is a commonly used metric in machine learning to evaluate the performance of a classification model. It measures the ratio of correctly predicted data samples to the total number of data samples. The accuracy score is a simple and intuitive way to assess how well a model is performing, but it may not always

be the best metric to use, especially when classes are imbalanced. The formula for accuracy is:

$$Accuracy = \frac{Number\ of\ Correct\ Predictions}{Total\ Number\ of\ Predictions}. \tag{2.9}$$

- Precision: Precision is another commonly used metric in the context of binary classification problems. It measures the ratio of true positive predictions to the total predicted positives. True positives are the input data correctly predicted as belonging to the positive class. False positives are the input data incorrectly predicted belonging to the positive class; they actually belong to the negative class. The choice of which of the two classes is positive or negative is up to the user. The formula is given by:

$$Precision = \frac{True\ Positives}{True\ Positives\ +\ False\ Positives}. \tag{2.10}$$

- Mean Squared Error (MSE): Mean squared error is not used only as a loss function, but it is also used as a regression metric. It is used to quantify the difference between the predicted values and the actual outputs. The formula remains the same (2.4).

- Root Mean Square Error: Root mean square error is not used only as a loss function, but it is also used as a regression metric. It is used to quantify the square root of the MSE. The formula remains the same (2.5).

- Mean Absolute Error (MAE): Mean absolute error is also used as a regression metric, not only as a loss function. It is used to quantify the absolute difference between the predicted values and the actual outputs. The formula remains the same (2.6).

# Chapter 3

# Non-Linear Problems

*The goal of this chapter is to introduce the concept of a non-linear problem and to explain the main machine learning approaches to address it. It starts by describing the concept of feature mapping, then it illustrates the main characteristics of kernels, random features and neural networks; the most common techniques to address non-linearity.*

## 3.1 Feature mapping

Non-linearity is a concept often encountered in modern machine learning problems. A non-linear problem is a problem whose data have been generated through a non-linear process, so these data have non-linear relationships. Non-linear classification refers to the situation where input data points from different classes cannot be classified using a linear function because the underlying patterns or relationships are too complex. In other words, a single straight line cannot accurately divide the data points. Non-linear regression involves predicting a continuous target variable based on input features where the relationship between the inputs and the target is non-linear. In simpler terms, a single straight line does not fit the data points well. To address non-linear problems, non-linear models are required. Feature mapping is a way to build non-linear models. It involves the process of transforming the original feature space of input data to better represent the underlying patterns and relationships in the data. There exist two types of feature mapping:

- Linear Feature Mapping: It involves creating new features by applying linear operations (e.g., addition, subtraction, and scaling) to the existing features in a dataset. Linear feature mapping does not change the dimensionality of the original feature space. This technique can help capture linear patterns and relationships between

features. It is often used when the underlying patterns and relationships in the data are linear.

- Non-Linear Feature Mapping: This technique transforms the original feature space into a higher-dimensional feature space, by creating and adding new features that are typically derived from the original features. For example, when working with a dataset containing two features represented as $x=(x_1,x_2)$, we can generate an additional feature $x_3 = (\frac{x1}{x2})$, and subsequently reshape the original input data to $x = (x_1, x_2, x_3)$. This transformation elevates the original 2D data to a 3D representation and it allows us to learn non-linear relationships using a linear model on a higher dimensional space.

One approach to tackle non-linear problems in machine learning is by stacking a linear model on top of a non-linear feature mapping, thereby combining a problem which can be solved easily (linear models are easy to solve), with higher representation capabilities. Consult Chapters 3, 7 and 9 of the book [JWH$^+$23] for additional information. However, manually defining feature mappings is cumbersome and not easy to do automatically (unless one has special knowledge of the specific type of features which should be added). In the next section, we will look at kernel methods which allow us to define an *infinite dimensional* feature map, which can still be solved easily.

## 3.2 Kernels

An infinite-dimensional kernel mapping function, or simply kernel, is a function $k : X, X \rightarrow R$ that given two input data, returns a real number. The kernel matrix is a symmetric matrix $K_n$ of $n$ rows and columns, where $n$ is the number of input data. Each entry in the kernel matrix is the result of applying the kernel function to the corresponding pair of input data. This means that the element $K_{i,j} = k(x_i, x_j)$. The kernel matrix encapsulates the pairwise similarities between all the input data. It enables machine learning models to operate in the higher-dimensional feature spaces without computing explicitly the transformed feature vectors. Given the input data $X = (x_1, ..., x_n)$, the associated labels $Y = (y_1, ..., y_n)$, the identity matrix $I$ and a positive hyperparameter $\lambda$ which can be fine-tuned with cross-validation, a set of coefficients $c = (c_1, ..., c_n)$ can be computed by resolving the following linear system $c = (K_n + n\lambda I)^{-1}Y$. This linear system is derived from the objective function of kernel ridge regression: $min_c||K_nc - Y||^2 + \lambda n||c||^2$. The term $||K_nc - Y||^2$ is the squared loss, while $\lambda n||c||^2$ is the regularization term. Given the coefficients $c$ and a symmetric, positive definite kernel $k(x_i, x)$ that builds a positive

definite kernel matrix, a non-linear estimator function can be built as follows:

$$\hat{f}(x) = \sum_{i=1}^{n} k(x_i, x)c_i. \tag{3.1}$$

Common symmetric and positive definite kernels are the following:

- Linear Kernels: A linear kernel performs a linear feature mapping of the data. Given two input data points $x$ and $x_i$, the linear kernel is defined as the dot product of them:

  $$k(x_i, x) = x_i^\top x. \tag{3.2}$$

  The primary advantage of using a linear kernel is its simplicity. It is well-suited for linearly separable datasets because it transforms the data without changing the dimensionality to help the models capture linear patterns and relationships.

- Polynomial Kernels: Polynomial kernels are designed to capture non-linear patterns and relationships in the data by computing the polynomial expansion of the input data. They perform a non-linear feature mapping. Given two input data points $x$ and $x_i$, a positive integer $d$ that determines the degree of the polynomial transformation and the "width" of the kernel and a constant $c$ that can be adjusted to control the offset of the polynomial transformation, the polynomial kernel is defined as:

  $$k(x_i, x) = (x_i^\top x + c)^d. \tag{3.3}$$

  Polynomial kernels are well-suited for non-linearly separable datasets; they increment the dimensionality of the data to help the models capture more complex non-linear patterns and relationships.

- Radial Basis Function Kernels: Also known as Gaussian kernels, radial basis function (RBF) kernels are designed to capture non-linear patterns and relationships in the data performing a non-linear feature mapping. Given two input data points $x$ and $x_i$, the Euclidean distance $||x_i - x||$ between the two input data points, a positive hyperparameter $\sigma$ that determines the "width" of the kernel, the RBF kernel is defined as:

  $$k(x_i, x) = e^{(-\frac{||x_i - x||^2}{2\sigma^2})}. \tag{3.4}$$

  A smaller value of $\sigma$ makes the kernel wider, while a larger value makes it narrower. Selecting an appropriate value for the $\sigma$ hyperparameter is crucial; an incorrect choice can deeply impact the performance. RBF kernel is a positive-definite kernel; applying it to all input data points ensures that the corresponding kernel matrix is positive and semi-definite, making it very suitable for machine learning models.

- Laplacian Kernels: Also known as Laplacian RBF kernels, are kernels designed to capture non-linear patterns and relationships in the data performing a non-linear feature mapping. Given two input data points $x$ and $x_i$, the Euclidean distance $||x_i - x||$ between the two input data points, a positive hyperparameter $\sigma$ that determines the "width" of the kernel, the Laplacian kernel is defined as:

$$k(x_i, x) = e^{(-\frac{||x_i - x||}{\sigma})}. \tag{3.5}$$

  Laplacian kernels are particularly useful when data may exhibit discontinuities. They are more robust to outliers.

Consult Chapter 9 of the book [JWH+23] for additional information about kernels.

## 3.3 Random features

Random features are a technique employed in machine learning to approximate kernels through the use of randomization. This approach reduces the computational costs, making it particularly valuable when working with large datasets or high-dimensional feature spaces. This is because the size of the kernel matrix scales with the square of the number of data points. So, if we have $n$ input data points, the cost of construction of the kernel matrix is $O(n^2)$. This means that high-dimensional feature spaces have no particular impact on the computation cost of kernel ridge regression (KRR). The infinite-dimensional kernel mapping function is approximated via a finite-dimensional randomized feature mapping function $z(x)$. If the dimension of this feature mapping function is lower than the number of data points, the problem can be solved in a smaller amount of time than the original kernel problem. The function $z(x)$ makes also use of weights and biases. These parameters are usually generated randomly according to a certain distribution. Given the input data $X = (x_1, ..., x_n)$, the associated labels $Y = (y_1, ..., y_n)$, the identity matrix $I$ and a positive hyperparameter $\lambda$ which can be fine-tuned with cross-validation, a set of coefficients $c = (c_1, ..., c_n)$ can be computed by resolving the following linear system $c = (Z_X^\top Z_X + n\lambda I)^{-1} Z_X^\top Y \approx (K_n + n\lambda I)^{-1} Y$. This linear system is derived from the objective function of kernel ridge regression in terms of $Z_X$: $min_c ||Z_X^\top Z_X c - Y||^2 + \lambda n ||c||^2$. $X$ is the $(n, d)$ matrix of input data, while $Z_X$ is the $(n, e)$ random features matrix, where $e$ is the number of random features. Given two input data points $x$ and $x_i$ with $i = 1, ..., n$ where $n$ is the number of input data points, the randomized feature mapping function $z(x)$ approximating a kernel function $k(x_i, x)$ and a set of coefficients $c$, the estimator function (Formula 3.1) becomes:

$$\hat{f}(x) = \sum_{i=1}^{n} k(x_i, x) c_i \approx \sum_{i=1}^{n} z(x_i)^\top z(x) c_i. \tag{3.6}$$

By applying the randomized feature mapping function $z(x)$ to each input data point, the original feature space turns into a higher dimensional feature space. Once the data has been transformed in this manner, a linear model can be used on top of the transformed data with lower computational complexity than the kernel ridge regression problem. Different random feature mapping functions approximate different kernel functions. We are interested in random ReLU features and random Fourier features.

- Random ReLU Features: To generate random ReLU features, the randomized feature mapping function $z(x)$ is the common ReLU activation function used in neural networks which is the topic of the next section. So, given $r$ the dimensionality of $z(x)$ and the vectors of weights $w = (w_1, ..., w_r)$ and biases $b = (b_1, ..., b_r)$, the term $z(x_i)^\top z(x)$ is computed as:

$$z(x_i)^\top z(x) = \frac{1}{r}\sum_{j=1}^{r}(max(0, w_j^\top x_i + b_j))(max(0, w_j^\top x + b_j)). \tag{3.7}$$

  The vectors of weights and biases are usually generated through the Gaussian distribution or the continuous uniform distribution over the surface of a sphere in $d$ dimensions, or uniform distribution on the hypersphere. According to this last distribution, given a sphere of radius $R$, the probability density function (PDF) is expressed as $f(x) = \frac{R}{\sigma_d}$, where $\sigma_d = \frac{R}{S_d}$, $S_d = \frac{2\pi^{d/2}}{\Gamma(d/2)}$ and $\Gamma()$ is the gamma function; an extension of the factorial function to real and complex numbers. Random ReLU features are similar to a 1-hidden layer neural network with a ReLU activation function, where the first-layer weights are random and not trained. It has been shown that random ReLU features approximate the arc-cosine kernel [SGT19].

- Random Fourier Features: The common Gaussian kernel is approximated by random Fourier features introduced in [RR07]. Given $r$ the dimensionality of the randomized feature mapping function $z(x)$ and the vectors of weights $w = (w_1, ..., w_r)$ and biases $b = (b_1, ..., b_r)$, the term $z(x_i)^\top z(x)$ is computed as:

$$z(x_i)^\top z(x) = \frac{1}{r}\sum_{j=1}^{r}(cos(w_J^\top x_i + b_J), sen(w_j^\top x_i + b_J))(cos(w_j^\top x + b_j), sen(w_j^\top x + b_j)). \tag{3.8}$$

  The vectors of weights and biases are generated through a Gaussian distribution $p(\omega)$ and the uniform distribution $[0, 2\pi]$ respectively.

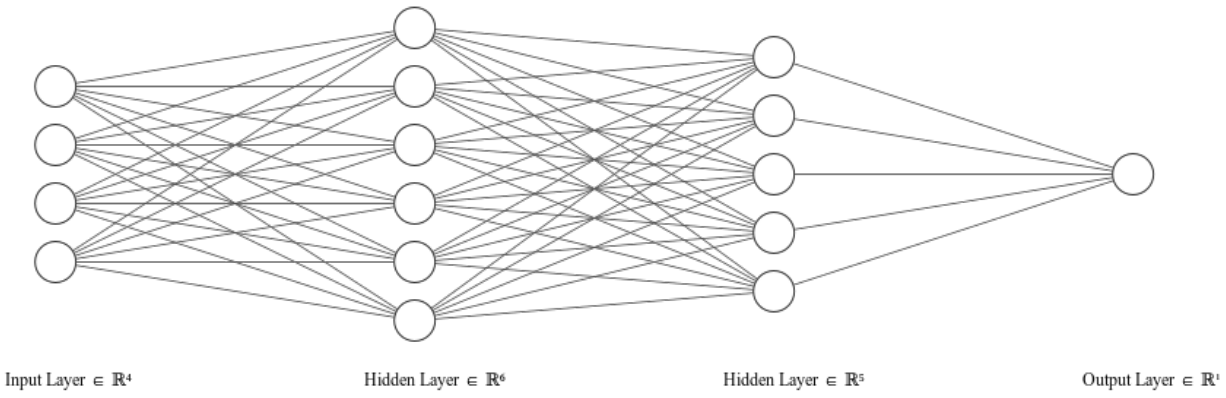
## 3.4 Neural networks

Neural networks, often referred to as artificial neural networks (ANNs), are a class of non-linear machine learning models inspired by the structure and function of the human

brain. Neural networks have gained popularity in recent years due to the wide range of applications and their ability to learn complex patterns and features from data. Neural networks learn feature mappings implicitly during the training process. A neural network is composed of neurons, which are also referred to as nodes or units. Neurons are linked with weighted connections, and each connection has an associated bias. These weights and biases are learned during the training process and determine how information flows through the network. Neural networks are organized into layers of neurons. As shown in Figure 3.1, the most common types of layers in a neural network include:

- Input Layer: This layer stores the input data and passes them to the subsequent layer.
- Hidden Layer: These are one or more layers in between the input and output layers. Each neuron in a hidden layer performs a computation on the received input, and the resulting output is passed as input to the neurons of the subsequent layer, according to the connections. Figure 3.1 shows an example of a neural network with two hidden layers. Neural networks with many hidden layers are called deep neural networks (DNNs). DNNs are typically overparametrized.
- Output Layer: This is the final layer which produces the prediction of the neural network.

Figure 3.1: An example of neural network.




Neurons, excluding the ones in the input layer, apply an activation function to their inputs to capture non-linear and complex patterns or relationships. Common activation functions include:

- Rectified Linear Unit (ReLU): It is one of the most popular hidden layer activation functions. Given an input data point $x$, the formula is defined as:

$$\sigma(x) = max(0, x). \tag{3.9}$$

It replaces negative input values with zero and leaves positive values unchanged. This makes it computationally more efficient than functions like sigmoid and hyperbolic tangent because it involves a simple thresholding operation. ReLU helps mitigate the vanishing gradient problem, which can occur in deep neural networks with activation functions like sigmoid and tanh. The gradient of ReLU is 1 for positive values, which can allow for more efficient gradient flow during backpropagation.

- Sigmoid: The sigmoid activation function is non-linear; it exhibits an S-shaped curve. Given an input data point $x$, the formula is as follows:

$$\sigma(x) = \frac{1}{1 + e^{-x}}. \tag{3.10}$$

This function has an output range between 0 and 1. It is often used in the output layer of neural networks. It is used almost for binary classification tasks. In hidden layers, the sigmoid function has been largely replaced by the ReLU and its variants due to their ability to mitigate the vanishing gradient problem and accelerate training in deep neural networks.

- Hyperbolic Tangent (tanh): The hyperbolic tangent function is a non-linear activation function that exhibits an S-shaped curve. Given an input data point $x$, the formula is as follows:

$$\sigma(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}. \tag{3.11}$$

This function has an output range between -1 and 1. This makes it zero-centered. The zero-centered property helps mitigate the vanishing gradient problem that can occur with the sigmoid activation function, especially in DNNs. The tanh activation function is often used in hidden layers of neural networks. It can be used for binary classification, multi-class classification and regression tasks.

- Softmax: The softmax activation function is commonly used in the output layer of neural networks for multi-class classification problems. Given $x_i$ the i-th component of the input vector and $N$ the number of components in $x_i$, the formula is defined as:

$$\sigma(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}} \tag{3.12}$$

The softmax function is often used in the output layer of a neural network for multi-class classification problems. The class with the highest probability is the predicted class for the input.

Given the input data $(x_1, ..., x_n)$, the weights $(w_1, ..., w_n)$ associated with the input data, an activation function $\sigma$ and a bias term $b$, which is a constant that allows to shift the activation function to the left or right, the output of a single neuron with $n$ connection is computed as:

$$y = \sigma(\sum_{i=1}^{n}(w_i x_i) + b). \tag{3.13}$$

The training process of a neural network, where the network learns to make accurate predictions by adjusting its weights and biases, comprises three phases:

1. Initialization: Initially, the parameters of the model (weights and biases) are set to some initial values.

2. Forward Propagation: It is the initial step in which the neural network processes input data and generates predictions based on the current values of the parameters. In this phase, the neural network receives and stores input data in the input layer. Then, depending on the connections, neurons in the successive layers produce outputs according to Formula 3.13, which are passed as inputs to the neurons in the next layers. The prediction of the output layer is subject to evaluation by the backpropagation phase.

3. Backpropagation: The goal of backpropagation is to adjust the parameters to minimize the loss between the prediction and the actual output. Backpropagation proceeds as follows:

   3.1 Loss Calculation: The loss between the predicted output and the actual output is computed.

   3.2 Gradient Calculation: For each layer in the neural network, moving back from the output layer to the input layer, the chain rule is applied to compute the gradients of the loss concerning the parameters of the current layer. These gradients are also called "local gradients" and indicate how sensitive the loss is to changes in the parameters of that specific layer. The local gradients for each layer are computed by combining the gradients from the current layer and the gradients from the subsequent layer. The gradients are accumulated and used to adjust the parameters of each layer.

4. Optimization Algorithms: Optimization algorithms (e.g., SGD, GD, or Adam) use these gradients to adjust the parameters to minimize the loss.

5. Iteration: Steps 2 to 4 are repeated until the loss converges to a minimum value or after a specific number of iterations.

There are many types of neural networks, each designed for specific tasks and problems. Here are the most common types of neural networks:

- Feedforward Neural Networks (FNNs): Also known as multilayer perceptrons (MLPs), these are the simplest type of neural networks. They consist of an input layer, one or more hidden layers, and an output layer. They are used for tasks like classification and regression. The data flows forward from the input layer to the output layer through the hidden layer(s), without loops.

- Convolutional Neural Networks (CNN): CNNs are designed for processing grid-like data, such as images and videos. They consist of the following types of layers: one input layer, convolutional layers, pooling layers, activation layers, one flattening layer, fully-connected layers and one output layer. The input layer stores the image to be processed. A convolutional filter, or convolutional kernel, is a small matrix of weights. It is smaller than the input data and is designed to detect specific patterns or features in the image. Convolutional layers perform convolutional operations. A convolutional operation consists of sliding the convolutional filter over the input data and computing the dot product between the filter and the local region of the input it is currently covering. Pooling layers are applied periodically to reduce the dimensionality of the data to prevent overfitting and accelerate the computation. Activation layers apply activation functions on the outputs of convolutional layers. The flattening layer is used to convert the multi-dimensional data into one-dimensional vector. It is often set between a convolutional layer or a pooling layer and a fully-connected layer. Fully-connected layers take the output of a flattening layer and compute the final output which is stored in the output layer. CNNs are used for tasks related to image processing and computer vision.

- Autoencoders: An autoencoder is a type of neural network used for unsupervised learning and dimensionality reduction. Its structure is characterized by two components; an encoder network and a decoder network. The encoder comprises the input layer, one or more hidden layers whose number of neurons decreases with the depth. Its last hidden layer is called the "bottleneck" layer. This layer represents the compact representation of the input data. The decoder part starts with the bottleneck layer, zero or more hidden layers whose number of neurons gradually increases, and the output layer with the same number of neurons as the input layer. Autoencoders are useful for data compression, noise reduction and anomaly detection.

Consult the book [GBC16], Chapter 10 of the book [JWH+23] and the paper [Led21] for additional information about neural networks, their main types and their learning process.

# Chapter 4

# Theory Of Machine Learning

*This chapter discusses the standard theory and the modern theory. It first starts by discussing two important machine learning techniques: regularization and hyperparameter tuning. Then it explores the bias-variance tradeoff and the related concepts of underfitting and overfitting. Later, this chapter describes the double descent phenomenon; a new pattern to which modern machine learning models fit. In the end, it introduces some of the main contributions of Mikhail Belkin.*

## 4.1 Regularization

Regularization is a technique used in machine learning to improve the model performance on new data by adjusting its parameters during the training phase. The main regularization methods in machine learning are:

- L1 Regularization: L1 regularization, also known as Lasso (Least Absolute Shrinkage and Selection Operator) regularization, adds a penalty term to a loss function, which is proportional to the absolute values of the parameters of the model. Given $L$ the loss function, $\lambda$ the regularization constant that controls the strength of the L1 penalty, and $|\theta_i|$ the absolute value of the parameter $\theta_i$ of the model, the modified loss function is as follows:

$$L' = L + \lambda \sum_i |\theta_i|. \tag{4.1}$$

  The L1 regularization penalty encourages the optimization algorithm to drive some of the parameters $\theta_i$ to zero. As a result, features associated with these zero coefficients are effectively excluded from the model.

- L2 Regularization: L2 regularization, also known as Ridge regularization, also works by adding a penalty term to a loss function, which is proportional to the squared values of the parameters of the model. Given $L$ the loss function, $\lambda$ the regularization constant that controls the strength of the L2 penalty, and $\theta_i^2$ the squared value of the parameter $\theta_i$ of the model, the modified loss function is as follows:

$$L' = L + \lambda \sum_i (\theta_i^2). \tag{4.2}$$

  The L2 regularization penalty encourages the optimization algorithm to reduce the magnitude of the coefficients of the model. This has the effect of discouraging features from having an excessively large impact on the predictions of the model.

- Dropout: In a particular class of machine learning models called neural networks, dropout is a regularization technique where randomly selected units, or neurons, are ignored or "dropped out" during training.

Consult Chapter 6 of the book [JWH+23] for additional information.

## 4.2 Hyperparameter tuning

Hyperparameter tuning in machine learning is the process of finding the optimal set of hyperparameters for a machine learning model during the validation phase to improve its performance on a specific task. This process consists of the following key steps and techniques:

1. Define the Hyperparameters: This phase identifies the hyperparameters of the machine learning model to tune.

2. Select a Search Space: It defines a search space for each hyperparameter. This means specifying the range of possible values that each hyperparameter can take. The search space can be continuous (e.g., a range of values) or discrete (e.g., a list of options).

3. Search Method: Hyperparameter search methods involve systematically exploring a range of hyperparameter values to determine the combination that yields the best model performance. The hyperparameters identified by the search methods are then used by the cross-validation.

4. Cross-Validation: Cross-validation is a strategy that helps to assess the performance of the model for different hyperparameters by splitting the training set into multiple subsets for auxiliary training and evaluation of the model performance. The

assessment utilizes the same loss function as the primary training phase. It helps to improve model performance on new data because the model is trained and evaluated on different subsets of data. It can be computationally expensive, especially with a large number of folds or when used with complex models. The results of cross-validation may vary depending on the random splitting of data, which can make it less reproducible. Common cross-validation techniques include k-fold cross-validation, leave-one-out cross-validation, and hold-out validation. The choice of cross-validation method depends on the specific problem and dataset.

5. Early Stopping: Early stopping stops the training process when the performance of the model starts deteriorating.

The hyperparameter tuning process does not guarantee a significant improvement in model performance. Consult Chapter 5 of the book [JWH+23] and the paper [YS20] for additional information about hyperparameter tuning.

## 4.3 Bias-variance tradeoff

In machine learning, the generalization error is a measure of how well a machine learning model makes predictions on data it has not seen during the training phase; the test set. The bias-variance tradeoff is a theory that describes the relationship between the complexity of a machine learning model and its generalization error. The generalization error is calculated as the sum of two factors: the bias and the variance. Consider we wish to model a function $y = f(x) + \epsilon$, given a training set of $n$ samples $S = (x_1, y_1), \ldots, (x_n, y_n)$ drawn independent and identically distributed from an unknown probability distribution $D$. The noise $\epsilon$ also comes from an unknown probability distribution $D'$ which has mean zero (i.e. $\mathbb{E}[\epsilon] = 0$). Given the function learned using this finite training set by $\hat{f}_S$ and the loss function through which we will measure the error of our model to be the squared loss, the generalization error which gives a complete measure of the error of $\hat{f}$ is as follows:

$$\mathbb{E}_{x,y,S}[(y - \hat{f}(x))^2].$$

The expectation is taken over the test input data point $x \sim D$, $y$ whose randomness, given $x$, comes from $\epsilon$ and the training set $S$ used to learn $\hat{f}_S$ (integrate over all possible training sets, coming from the distribution $D$). Fixed a test input data point $x$, the prediction of the algorithm averaged over all training sets is denoted by $\hat{f}(x) = \mathbb{E}_S[\hat{f}_S(x)]$. Also denote the averaged target variable $\bar{y} = \mathbb{E}_y[y] = \mathbb{E}_\epsilon[f(x) + \epsilon]$. We can express the bias-variance decomposition for a fixed $x$ as follows:

$$\begin{aligned}\mathbb{E}_{y,S}[(\hat{f}_S(x) - y)^2] &= \mathbb{E}_{y,S}[(\hat{f}_S(x) - \hat{f}(x) + \hat{f}(x) - y)^2] \\ &= \mathbb{E}_S[(\hat{f}_S(x) - \hat{f}(x))^2] + \mathbb{E}_y[(\hat{f}(x) - y)^2] + 2\mathbb{E}_{y,S}[(\hat{f}_S(x) - \hat{f}(x))(\hat{f}(x) - y)]\end{aligned}$$

where we have simply added and subtracted the average predictions $\hat{f}(x)$. The first term of this equation is the variance of our estimator $\hat{f}_S(x)$. The last term equals zero since $\mathbb{E}_{y,S}[(\hat{f}_S(x) - \hat{f}(x))(\hat{f}(x) - y)] = \mathbb{E}_S[\hat{f}_S(x) - \hat{f}(x)]\mathbb{E}_y[\hat{f}(x) - y]$ and $\mathbb{E}_S[\hat{f}_S(x) - \hat{f}(x)] = \mathbb{E}_S[\hat{f}_S(x)] - \hat{f}(x) = 0$ by definition of $\hat{f}(x)$. Given this, the bias must reside in the second term. Once again adding and subtracting $\bar{y}$ this time, the equation becomes:

$$\begin{aligned}\mathbb{E}_y[(\hat{f}(x) - y)^2] &= \mathbb{E}_y[(\hat{f}(x) - \bar{y} + \bar{y} - y)^2] \\ &= \mathbb{E}_y[(\hat{f}(x) - \bar{y})^2] + \mathbb{E}_y[(\bar{y} - y)^2] + 2\mathbb{E}_y[(\hat{f}(x) - \bar{y})(\bar{y} - y)].\end{aligned}$$

The first term is not random in $y$, so we can get rid of the expectation, and corresponds to the difference between the average predictions and the true target squared: the bias squared. The formula of the bias squared is as follows:

$$\mathbb{E}_y[(\hat{f}(x) - \bar{y})^2] = (\hat{f}(x) - \bar{y})^2 = \text{Bias}^2.$$

The second term is the deviation of $y$ around its mean: we can rewrite it to show that it corresponds to the variance of the noise $\epsilon$. This is an irreducible term which depends on the data (and not on the algorithm). The third term (cross-product) as before can be shown to be zero. Overall the error decomposition is as follows:

$$\begin{aligned}\mathbb{E}_{y,S}[(\hat{f}_S(x) - y)^2] =& \mathbb{E}_S[(\hat{f}_S(x) - \hat{f}(x))^2] \\ &+ (\hat{f}(x) - \bar{y})^2 \\ &+ \mathbb{E}_\epsilon[\epsilon^2].\end{aligned}$$

Bias is the difference between the average values of the predictions of the model and the actual outputs. This error term is introduced by approximating a real-world problem, which may be complex, by a simplified model. Given $y$ the actual output and $\hat{y}$ the prediction, the formula is as follows:

$$Bias(\hat{f}(x), y) = (\hat{f}(x) - \bar{y}). \tag{4.3}$$

Variance measures the variability of the predictions of a model. This error term is introduced by the sensitivity of the model to the variations in the training data. Given $\hat{y}$ the prediction, the formula is as follows:

$$Variance(\hat{f}(x)) = \mathbb{E}_S[(\hat{f}_S(x) - \hat{f}(x))^2]. \tag{4.4}$$

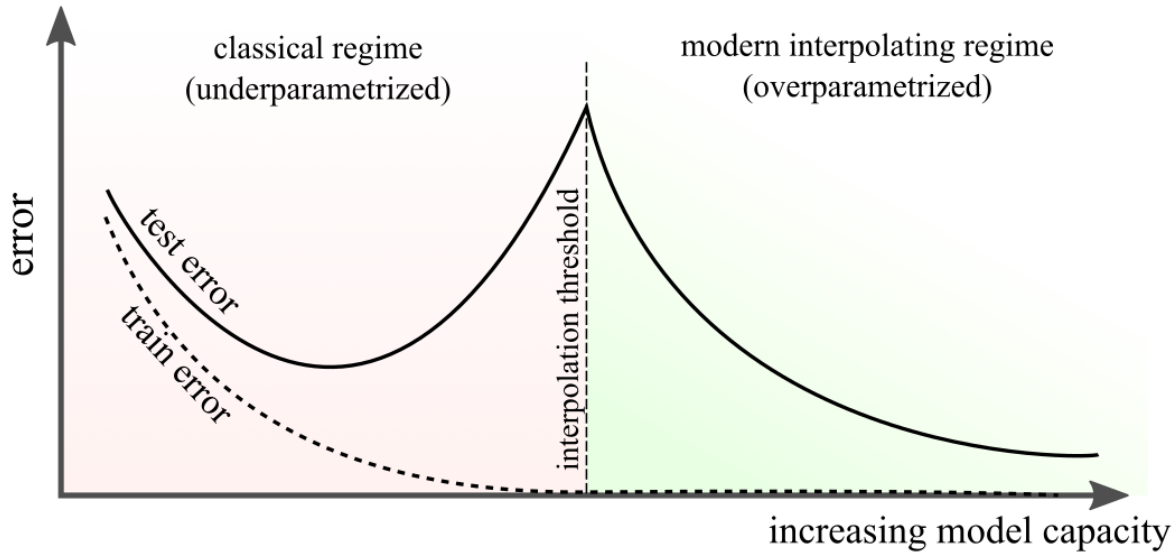
The complexity of a model is determined by the number of parameters, the range of values of the parameters, and the number of training samples. The relationship between model complexity and generalization error follows a U-shaped curve, with the generalization error decreasing as model complexity increases and then increasing again as the model becomes overly complex. Models with high bias and low variance usually tend to underfitting.

These models are not sensitive to the training data, so they do not capture the underlying patterns and relationships. This may lead to produce inaccurate predictions. Instead, models with low bias and high variance usually tend to overfitting. These models are highly sensitive to the training data such that they capture the noise and the variations in the data. This may lead to poor generalization of new data. The goal of machine learning is to strike a balance between bias and variance to build a model that fits well with new data. This involves choosing an appropriate model complexity and employing techniques like cross-validation for model hyperparameters and regularization for model parameters.

## 4.4 The double descent phenomenon

In recent years, a new phenomenon called "double descent" has been observed in various machine learning scenarios. Modern machine learning models rarely align with the bias-variance tradeoff, but they tend to have a new pattern as shown in Figure 4.1.

Figure 4.1: The double descent phenomenon.

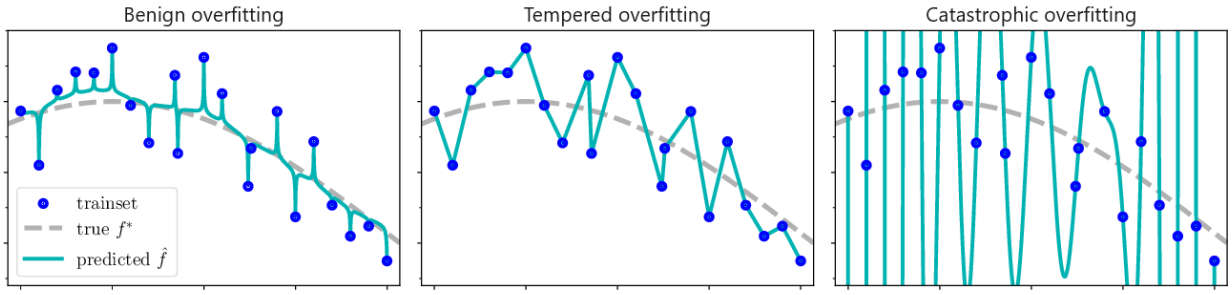


This modern theory splits the curve into two regions. In the first region, known as the classical regime, bias-variance regime or underparametrized regime, as the model complexity increases, the generalization error declines, and then rises. In the second region, known as the modern interpolating regime, modern regime or overparametrized regime, as the model complexity increases, starting from the interpolation threshold, the generalization error declines again. Further information is available in the book [JWH+23] and the paper [BHMM19].

## 4.5 Belkin contributions

Modern machine learning models tend to fit the double descent phenomenon, rather than the bias-variance tradeoff [BHMM19]. They also tend to interpolate the training data and perform well on new data and the type of loss function has no significant impact on the performance [BMM18]. Modern machine learning models, including kernels and ReLU neural networks which are neural networks with ReLu activation functions, tend to fit noisy data easily [BMM18]. Interpolated modern machine learning models can exhibit three types of overfitting [MSA+22] as shown in Figure 4.2:

Figure 4.2: Overfitting regimes.




- Bening Overfitting: Benign overfitting occurs when the interpolating methods approach the Bayes optimality even in the presence of noise when they are trained with large datasets. Models that approach the Bayes optimality mean that they make predictions that are as close as possible to the actual outputs.
- Catastrophic Overfitting: Catastrophic overfitting occurs when the interpolated models make predictions poorly. It is practically useless for real-world applications.
- Tempered Overfitting: Standard machine learning theory assumes that an interpolated model catastrophically overfits. Most interpolating methods do not overfit

benignly or catastrophically, but they show tempered overfitting which can be seen as a middleway between benign overfitting and catastrophic overfitting.

These types of overfitting are qualitative since different models can have different loss functions, data, number of epochs and more. Recognizing the type of overfitting is crucial for building machine learning models that perform well on new data. Appropriate model complexity and techniques like cross-validation and regularization are the main methods to achieve the best performance in terms of overfitting.

# Chapter 5

# Experiments

*This chapter consists of conducting a series of experiments on neural networks, random ReLU features and random Fourier features to check some of the Belkin contributions about them and to discuss and validate some of our hypotheses, derived from our knowledge, about the expected behaviour of these machine learning techniques.*

## 5.1 Settings, data and hypotheses

As the first step, we defined the types of random features and the neural network architecture which are going to build our models to perform binary classification tasks. As random features, we decided to use random ReLU features and random Fourier features. The neural network comprises one input layer with 784 neurons, one hidden layer with several neurons ranging from 1 to 50000 and one output layer with one neuron. The hidden layer and the output layer respectively apply the ReLU activation function and the sigmoid activation function. The dataset used for the training of our models is a subset of MNIST data. This subset comprises only the input data labelled as '3' or '5'. Our models are built to binary classify these data correctly. The training set and test set respectively comprise $n = 11552$ and $n = 1902$ data samples, each one with $d = 784$ features. In addition, we added noise to copies of this subset of data. Noise is added by flipping some labels chosen randomly according to a certain probability. Low levels of noise mean labels are flipped with 1% of probability. In case of high levels of noise, labels are flipped with 10% of probability. The training error is the ratio of the number of wrong predictions to the total number of data points during the training phase. The test error is the ratio of the number of wrong predictions to the total number of data points during the test phase. The training loss is the mean loss between the predictions and actual outputs of the training set. The test loss is the mean loss between the predictions and actual outputs of the test set. Once the

machine learning models have been defined, before their training, we derived the following series of hypotheses about their expected behaviour based on our knowledge:

- First Hypothesis: The training error and the training loss always decline as the model complexity increases. Adding noise means slowing down the descents of the errors and the losses in the best case, and increasing the starting values of these descents and their value gaps between. The classical bias-variance rules never kick in.
- Second Hypothesis: Interpolation of the training data is achieved by increasing the model complexity. As the amount of noise increases, the complexity of the model necessary to interpolate the training data increases.
- Third Hypothesis: In the absence of noise, the test error and the test loss precipitate to a finite value and remain stable without rising as the model complexity increases, hence the "high-variance" regime has no effect. Models show benign overfitting.
- Fourth Hypothesis: In the presence of low levels of noise in the training set, the test error and the test loss behave similarly to the case of absence of noise; they decrease more slowly to a finite value and remain stable. Here, models still show benign overfitting.
- Fifth Hypothesis: High amounts of noise in the training set render the model unable to correctly learn the underlying function. However, unlike the classical bias-variance tradeoff, as models become very complex, the test error does not explode, but tends to remain stable: they exhibit tempered overfitting. The test loss instead tends to increase briefly to a finite value and remain stable.
- Sixth Hypothesis: In the presence of low levels of noise both in the training set and the test set, compared to the fourth hypothesis, the test error decreases more slowly and more briefly to a finite value from a higher starting value and remains stable. The test loss briefly increases to a finite value starting from a higher starting value and remains stable. Adding noise also to the test set does not reduce the test error and the test loss, but it makes the model more "confusing". Here, models show benign overfitting.
- Seventh Hypothesis: High amounts of noise both in the training set and test set make the model unable to correctly learn the underlying function. Compared to the fifth hypothesis, the test error increases to a finite value and remains stable, while the test loss significantly increases to a finite value from a higher starting value and remains stable. Adding high levels of noise also to the test set does not reduce the test error and the test loss, but it makes the model furthermore "confusing". Here, models show catastrophic overfitting.

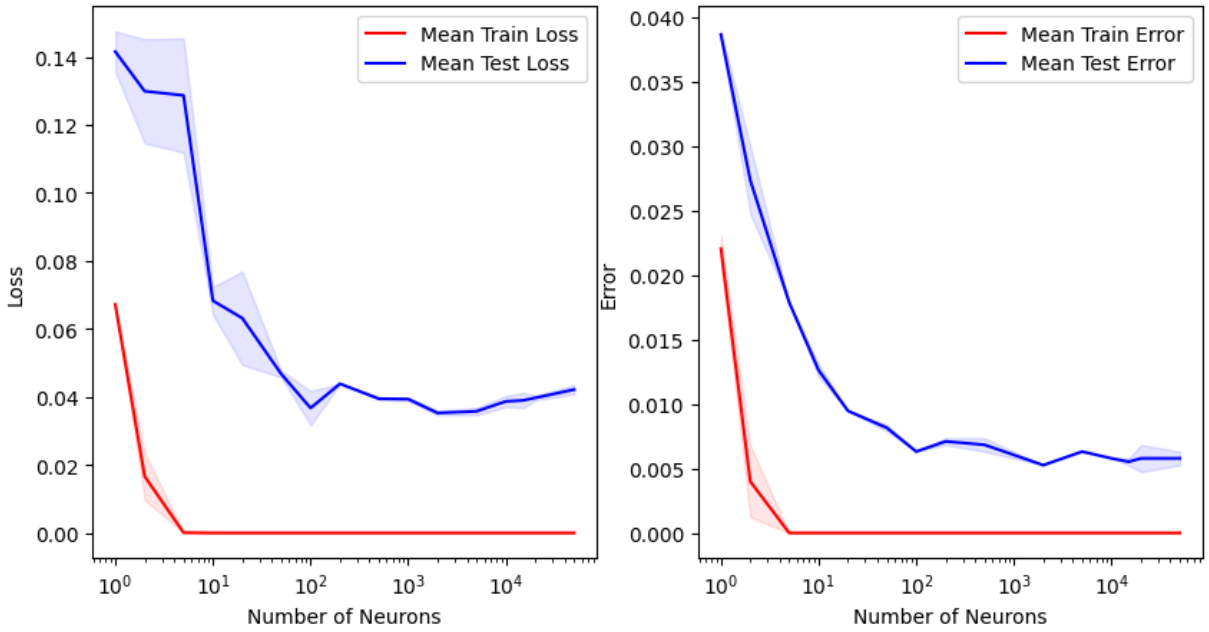
Neural networks are trained in 1000 epochs with binary cross-entropy loss function, Adam optimization algorithm with a learning rate $\gamma = 0.01$ and batches of 128 training samples each one. Models based on random ReLU features are trained with the same settings but with a learning rate $\gamma = 0.1$. Models based on random Fourier features are trained by resolving the linear system $c = (Z_X^\top Z_X)^{-1} Z_X^\top Y$ previously discussed. The regularization term $n\lambda I$ is absent. Their performance is analyzed using the MSE loss and 0-1 loss. The 0-1 loss applied to the entire dataset, as said before, is not only a representation of the mean loss of the model but also a representation of the mean classification error. The training process, along with the associated testing process, of each machine learning model has been performed ten times to compute the mean performance in ten runs along with their standard deviations. In particular, we will analyze the performance with increasing the complexity of the models, concerning only the last epoch, except for the random Fourier features models.

## 5.2 Experiments with neural networks

We executed the following series of experiments to analyze the performance of our neural network models and to verify our hypotheses:

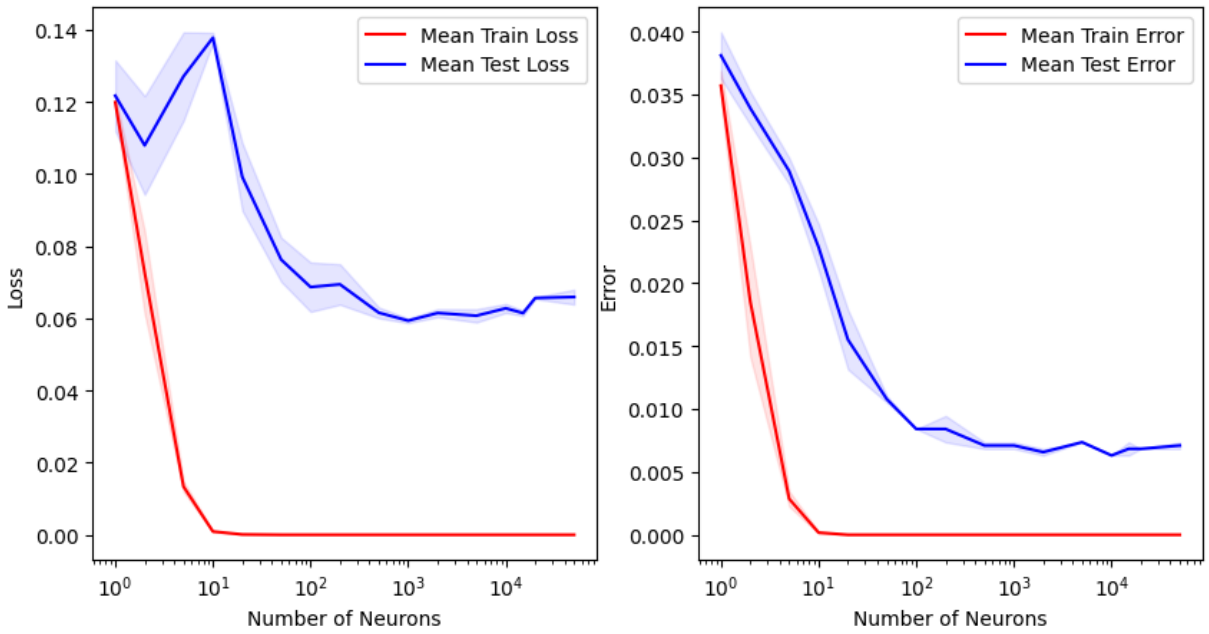
- First Experiment: Analyzing the performance of our neural network model in the absence of noise. As shown in Figure 5.1, as the model complexity increases, both the test error and the test loss decline and remain almost stable. This model does not align both with the bias-variance theory and the double descent theory. The train error starts decreasing from a value equal to about 0.020, while the train loss starts decreasing from a value equal to about 0.06. The test error starts decreasing from a value equal to about 0.040, while the test loss starts decreasing from a value equal to about 0.14. The model interpolates the training data at very low levels of complexity; with less than 10 neurons in the hidden layer. The model shows benign overfitting; the test error precipitates to a finite value and remains stable.

Figure 5.1: Mean losses and mean errors in the absence of noise.



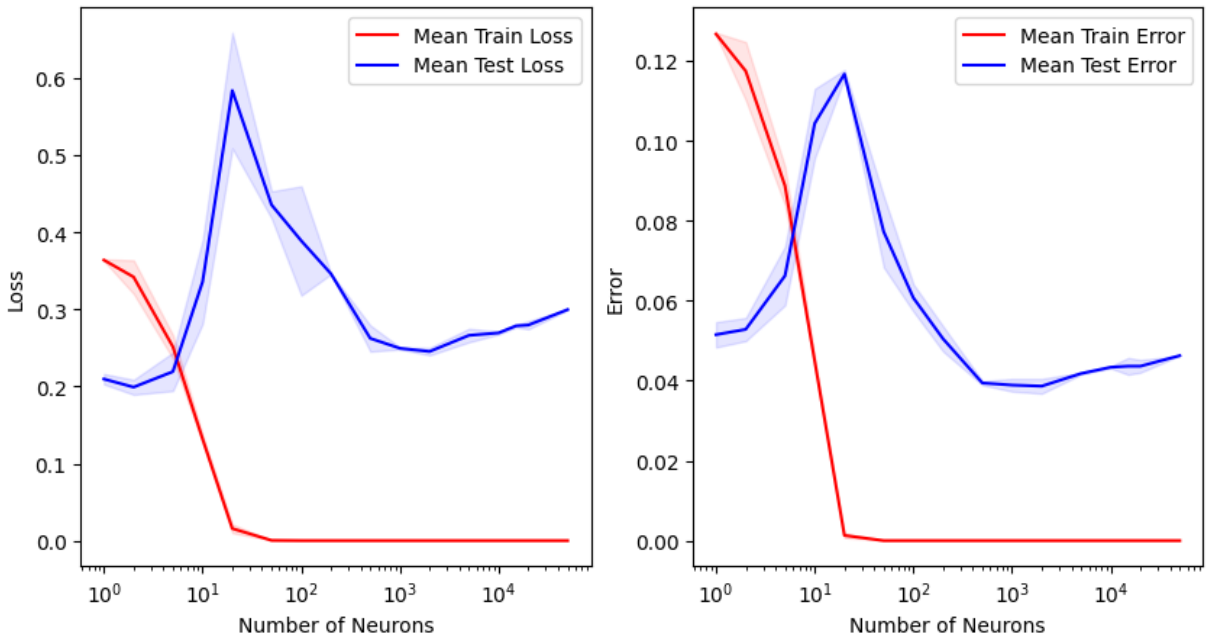

- Second Experiment: Analyzing the performance of our neural network model in the presence of 1% noise in the training set. As shown in Figure 5.2, as the model complexity increases, the test error declines and remains almost stable, while the test lost initially has a V-shape trend, then decreases and remains stable. This model does not align both with the bias-variance theory and the double descent theory. Compared with the previous case, increasing the levels of the noise up to 1% in the training set means slowing down the descents of the errors and the losses and increasing the interpolation threshold and the starting values of the training error descent and the training loss descent. The gaps between the errors and the losses are almost identical to the previous case. The starting values of the test error descent and the test loss descent are also almost identical. Here, the train error starts decreasing from a value equal to about 0.035, while the train loss starts decreasing from a value equal to about 0.12. The model interpolates the training data at a level of complexity strictly greater than 10 neurons in the hidden layer. The model shows benign overfitting; the test error decreases to a finite value and remains stable.

Figure 5.2: Mean losses and mean errors with 1% noise in the training set.



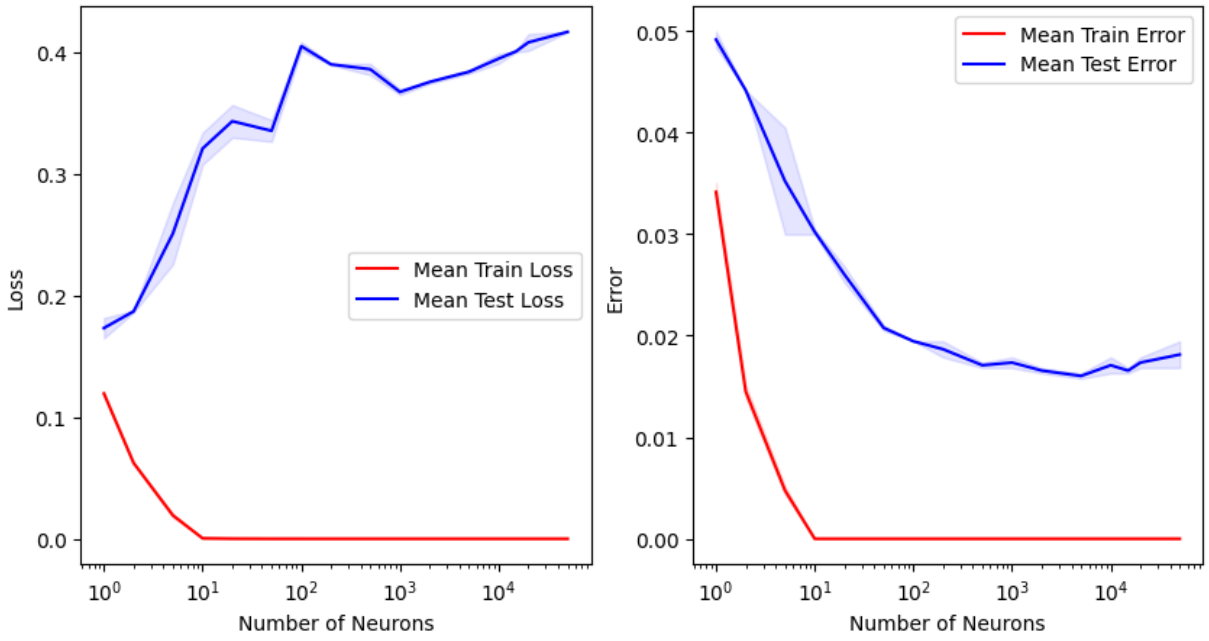

- Third Experiment: Analyzing the performance of our neural network model in the presence of 10% noise in the training set. As shown in Figure 5.3, as the model complexity increases, the test error and the test loss initially increase, then decline and remain almost stable. This model does not align both with the bias-variance theory and the double descent theory. As expected, compared with the previous cases, increasing the levels of noise up to 10% in the training set means slowing down the descents of the errors and the losses and increasing the starting values of these descents, the interpolation threshold and the gaps between the errors and the losses. The training error starts decreasing from a value equal to about 0.12 and remains stable. The training loss starts decreasing from a value equal to about 0.4 and remains stable. The test error starts decreasing from a value equal to about 0.12, while the test loss starts decreasing from a value equal to about 1.6. The model interpolates the training data at a level of complexity equal to about 100 neurons in the hidden layer. The model shows tempered overfitting; the test error increases, then decreases to a finite value similar to the starting one and remains stable.

Figure 5.3: Mean losses and mean errors with 10% noise in the training set.



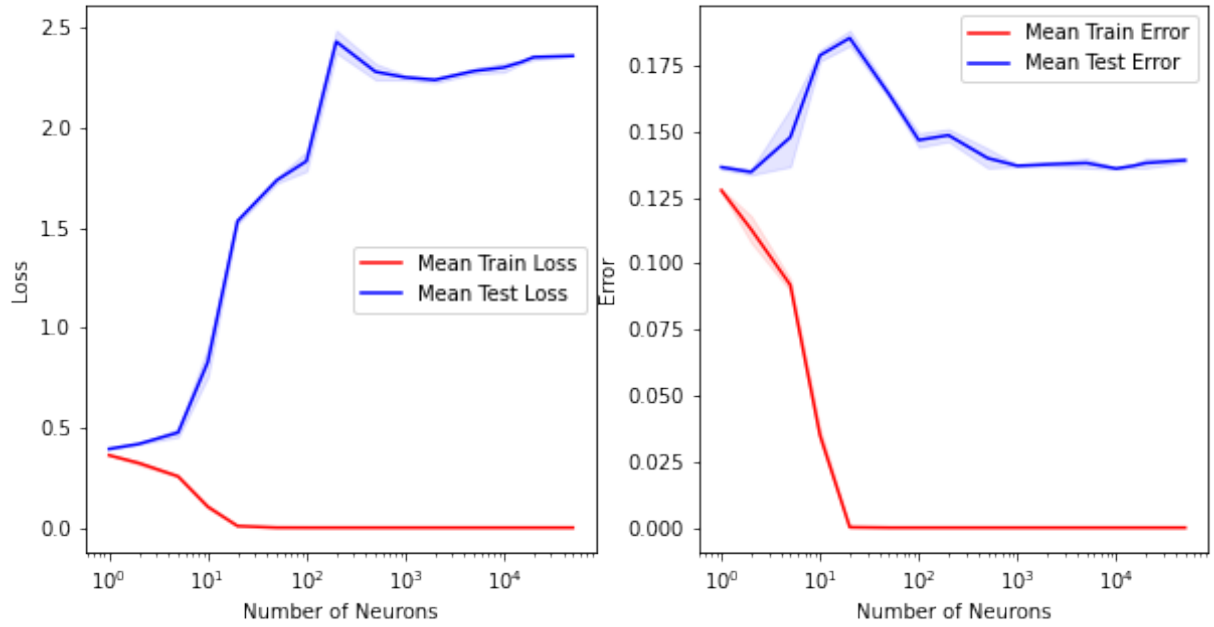

- Fourth Experiment: Analyzing the performance of our neural network model in the presence of 1% noise both in the training set and test set. As shown in Figure 5.4, as the model complexity increases, the test error declines and remains stable, while the test loss rises significantly. Adding noise in the test set too, means adding a sort of "confusion" to the model that increments with the model complexity. The more the model is "intelligent" and complex, the more is influenced by this concept of confusion. This model does not align both with the bias-variance theory and the double descent theory. Compared with the second experiment, the starting values of the training loss and the training error are almost identical. The interpolation threshold is also almost identical, but the starting values of the test loss and the test error are strictly greater. The descents of train loss and the train error are also almost equal, while the descent of the test error is slower. The model shows benign overfitting; the test error decreases to a finite value and remains stable.

Figure 5.4: Mean losses and mean errors with 1% noise both in the training set and test set.




- Fifth Experiment: Analyzing the performance of our neural network model in the presence of 10% noise both in the training set and test set. As shown in Figure 5.5, the behaviour of this model is similar to the previous experiment. As the model complexity increases, the test error briefly increases, then declines and remains stable, while the test loss rises significantly and remains stable. This model does not align both with the bias-variance theory and the double descent theory. Increasing the noise in the test set too, means increasing more the "confusion" of the model that increments with the model complexity. As expected, compared with all the previous cases, increasing the levels of noise up to 10% both in the training set and test set means slowing down the descents of the errors and the losses and increasing the starting values of these descents, the interpolation threshold and the gaps between the errors and the losses. The training error and the training loss both start decreasing from a value equal to about 0.125 and 0.5 respectively. The test performance got worse. The test error and the test loss both start decreasing from values equal to about 0.175 and 2.5 respectively. The model interpolates the training data about at the same level of complexity as the case of 10% noise in the training set. The model still shows tempered overfitting; the test error briefly increases, then concisely decreases to a finite value similar to the starting one and remains stable.

Figure 5.5: Mean losses and mean errors with 10% noise both in the training set and test set.



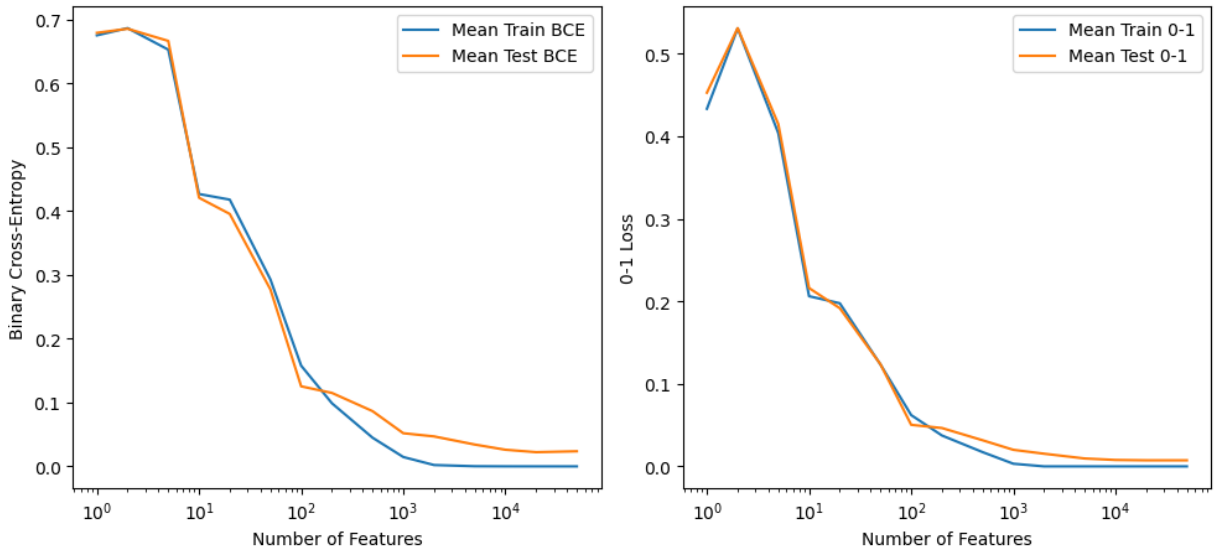

Contrary to the theory, from these experiments, we derived that our neural network models never align with the double descent theory. This means that the double descent phenomenon is not easily observable and very high levels of complexity are not sufficient to see it. Interpolation is achieved with low levels of complexity, even with significant quantities of noise. The presence of noise leads to a small increment of the interpolation threshold. Adding noise also implies slowing down the descents, increasing the starting values of these descents and the gaps between the losses and the errors. Our models demonstrated benign overfitting in most cases. All of these models interpolated the training data and performed well on test data, even in the presence of high levels of noise both in training data and test data. High levels of noise are not sufficient to lead to a catastrophic overfitting regime, so our neural network models are robust to the noise. These experiments confirm that modern machine learning models, including ReLU neural networks, tend to easily fit noisy data, interpolate the training data and perform well on new data. Contrary to the theory, these models do not tend to show tempered overfitting in most cases.

## 5.3 Experiments with random ReLU features

We performed a series of experiments to analyze the performance of our random ReLU features models and to verify our hypotheses. These experiments are the following:

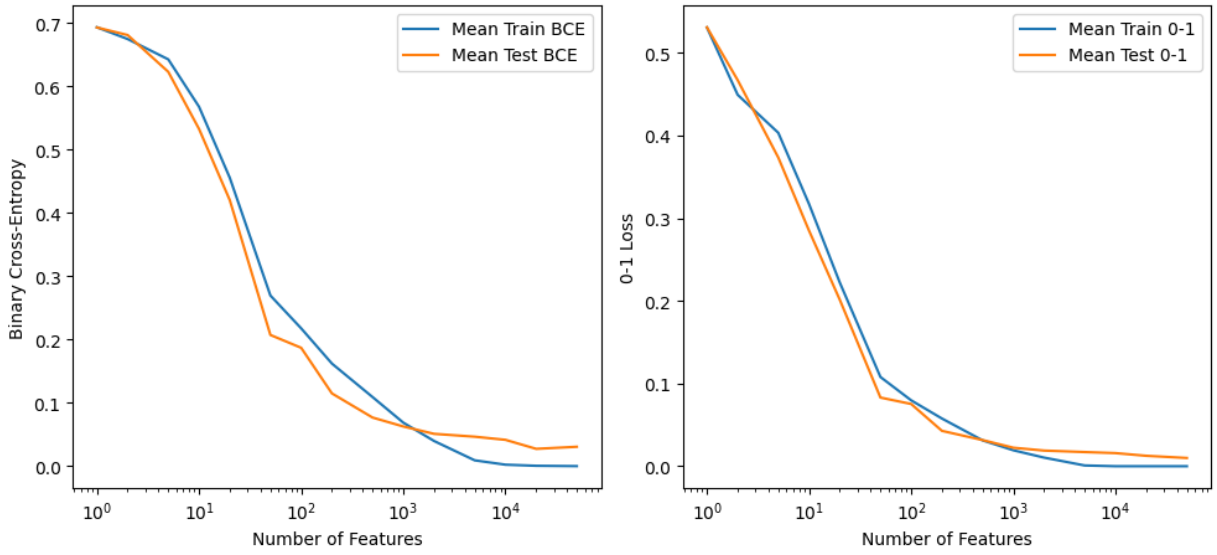
- First Experiment: Analyzing the performance of our random ReLU features model in the absence of noise. As shown in Figure 5.6, as the model complexity increases, the losses decline and remain stable, while the errors initially increase, then decline and remain stable. This model does not align both with the bias-variance theory and the double descent theory. There are no visible standard deviations. This means that this model performs uniformly through different runs. The 0-1 losses and the binary cross-entropy losses show similar trends. The errors start decreasing from a value equal to about 0.5. The losses start decreasing from a value equal to about 0.7. The model interpolates the training data at a level of complexity equal to about 1000 features. The model shows benign overfitting; the test error precipitates to a finite value and remains stable.

Figure 5.6: Mean losses and mean errors in the absence of noise.



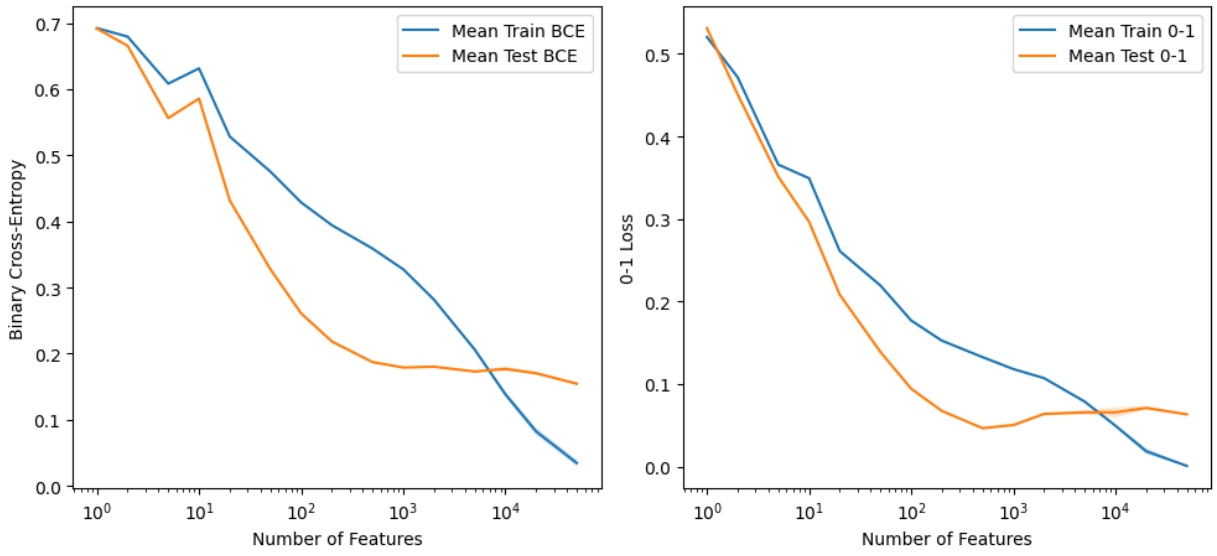
- Second Experiment: Analyzing the performance of our random ReLU features model in the presence of 1% noise in the training set. As shown in Figure 5.7, as the model complexity increases, both the losses and the errors decline and remain stable. This model does not align both with the bias-variance theory and the double descent theory. There are no visible standard deviations. This means that this model performs uniformly through different runs. The 0-1 losses and the binary cross-entropy losses show similar trends. Compared with the case of absence of noise, increasing the levels of the noise up to 1% in the training set slows down the descents of the errors and the losses and increases the interpolation threshold and the gap between the losses. The gap between the errors is very similar to the case of absence of noise, as the starting values. The model interpolates the training data at a level of complexity equal to about 10000 features. The model still shows benign overfitting; the test error precipitates to a finite value and remains stable.

Figure 5.7: Mean losses and mean errors with 1% noise in the training set.



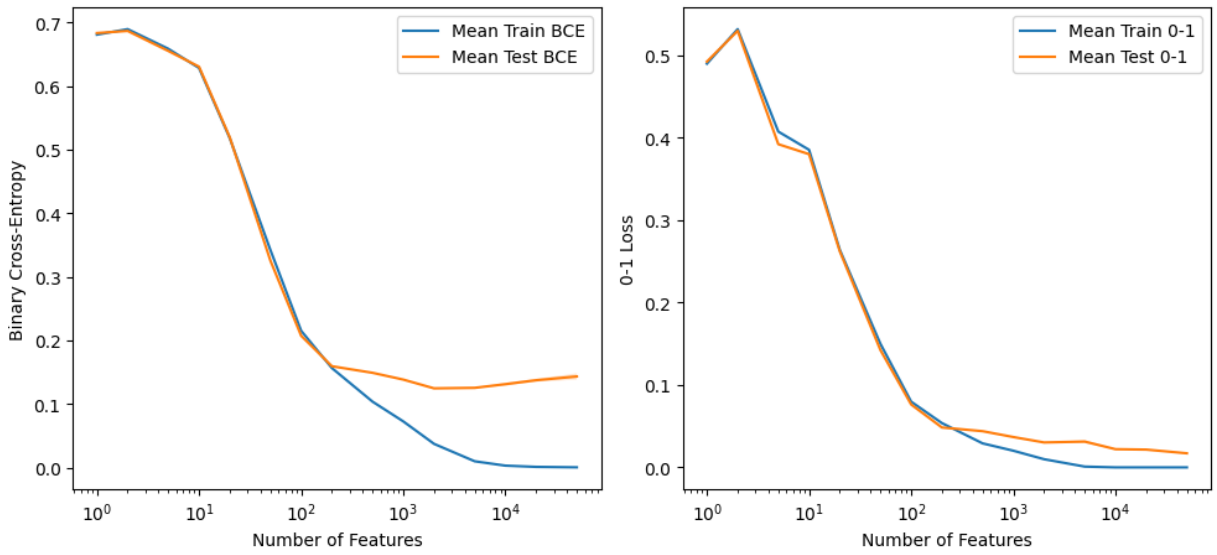
- Third Experiment: Analyzing the performance of our random ReLU features model in the presence of 10% noise in the training set. As shown in Figure 5.8, as the model complexity increases, both the losses and the errors decline. This model does not align both with the bias-variance theory and the double descent theory. There are no visible standard deviations. This means that this model performs uniformly through different runs. The 0-1 losses and the binary cross-entropy losses show similar trends. As expected, compared with the previous cases, increasing the levels of the noise up to 10% in the training set increases the slowdown of the descents of the errors and the losses, the gaps between the errors and the losses and the interpolation threshold. The starting values of these descents are also very similar to the previous experiments. The model needs more than 10000 features to interpolate the training data. The model shows benign overfitting; the test error decreases to a finite value and remains stable.

Figure 5.8: Mean losses and mean errors with 10% noise in the training set.



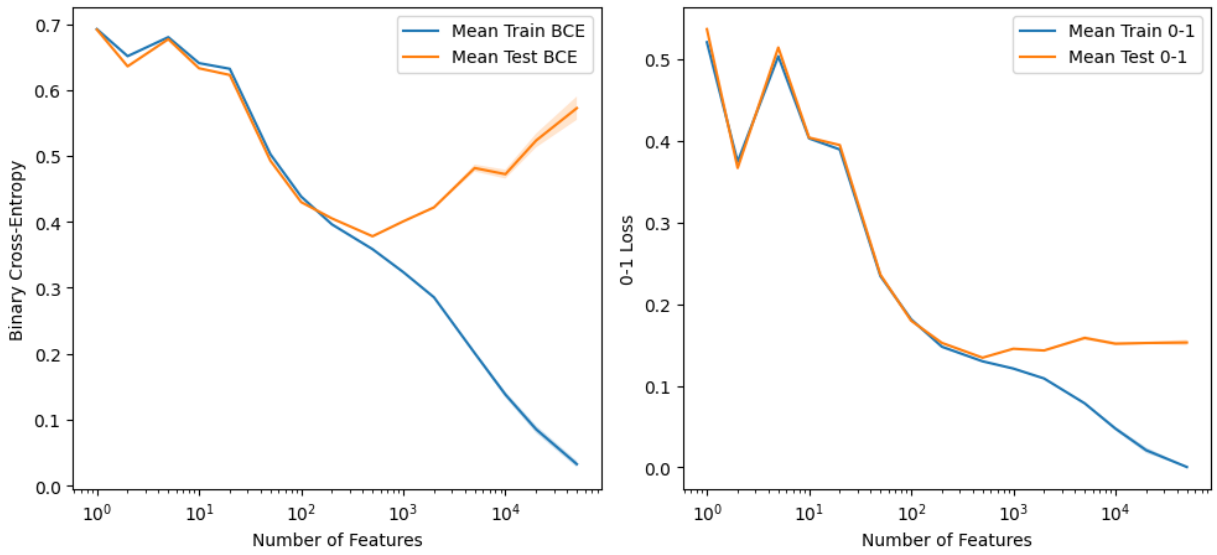
- Fourth Experiment: Analyzing the performance of our random ReLU features model in the presence of 1% noise both in the training set and test set. As shown in Figure 5.9, the behaviour of this model is similar to the second experiment. As the model complexity increases, the losses decline and remain stable, while the errors initially increase, then decline and remain stable. This model does not align both with the bias-variance theory and the double descent theory. There are no visible standard deviations. This means that this model performs uniformly through different runs. The binary cross-entropy losses show a greater gap than the 0-1 losses. Compared with the case of absence of noise, as in the second experiment, increasing the levels of the noise to 1% both in the training set and the test set slows down the descents of the errors and the losses and increases the interpolation threshold. Compared with both the first and second experiments, the gaps between the errors, and especially the losses, are more evident. Compared with the previous cases, the gap between the 0-1 losses, which is also the measure of the classification error, is lower than the gap between the binary cross-entropy losses. The starting values of these descents are very similar to the previous experiments. As in the second experiment, the model interpolates the training data at a level of complexity equal to about 10000 features. The model shows benign overfitting; the test error decreases to a finite value and remains stable.

Figure 5.9: Mean losses and mean errors with 1% noise both in the training set and test set.



- Fifth Experiment: Analyzing the performance of our random ReLU features model in the presence of 10% noise both in the training set and test set. As shown in Figure 5.10, as the model complexity increases, both the train loss and the train error decline. The BCE test loss briefly decreases, then increases, while the 0-1 test loss initially has a V-shaped trend, then decreases and remains stable. This model does not align both with the bias-variance theory and the double descent theory. There are no visible standard deviations. This means that this model performs uniformly through different runs. The binary cross-entropy losses show a greater gap than the 0-1 losses. As expected, compared with the previous experiments, increasing the levels of the noise up to 10% both in the training set and the test set slows down the descents of the errors and the losses and increases the gaps between the errors and the losses than any previous case. The starting values of these descents are similar to all the previous cases. Compared with the previous cases, the gap between the 0-1 losses, which is also the measure of the classification error, is lower than the gap between the binary cross-entropy losses. The starting values of these descents are also very similar to the previous experiment. The model needs more than 10000 features to interpolate the training data. The model shows benign overfitting; the test error decreases to a finite value and remains stable.

Figure 5.10: Mean losses and mean errors with 10% noise both in the training set and test set.



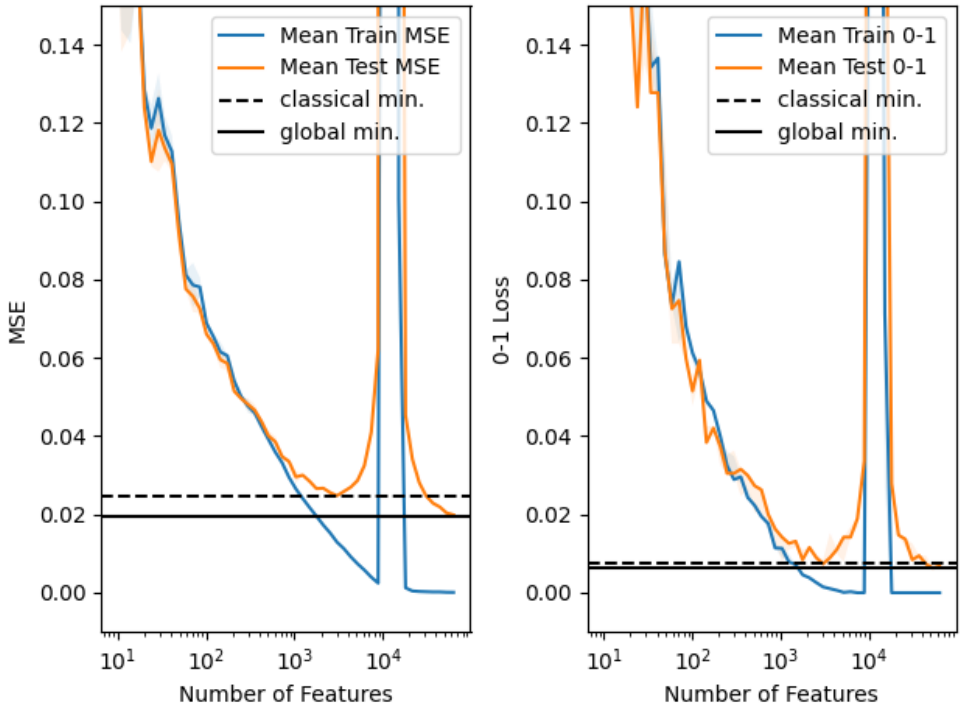
Contrary to the theory, these experiments demonstrate that our random ReLU features models never align with the double descent theory. This means that the double descent phenomenon is not easily observable and very high levels of complexity are not sufficient to see it. The presence of noise leads to a huge increment of the interpolation threshold, which needs basic significant levels of complexity. Increasing the levels of noise also implies slowing down the descents and increasing the gaps between the errors and the losses. The choice of the loss function has no significant impact on the performance of these models. In each model, the 0-1 losses and the binary cross-entropy losses show similar trends in most cases. The models show benign overfitting. Our models also tend to interpolate the training data and perform well on test data. High levels of noise are not sufficient to lead to a catastrophic overfitting regime, so these models are robust to the noise. These experiments confirm that most modern machine learning models, including random ReLU features models, tend to interpolate the training data and perform well on new data even in high amounts of noise and the choice of the loss function has no significant impact on the model performance. Contrary to the theory, these models do not tend to show tempered overfitting.

## 5.4 Experiments with random Fourier features

We performed also a series of experiments to analyze the performance of our random Fourier features models and to verify our hypotheses. In the following experiments, the classical minimum represents the minimum value of the test error achieved before the interpolation threshold, while the global minimum represents the global minimum value. Due to computational costs, we did not manage to show the convergence with model complexity of the test error of some experiments. These experiments are the following:

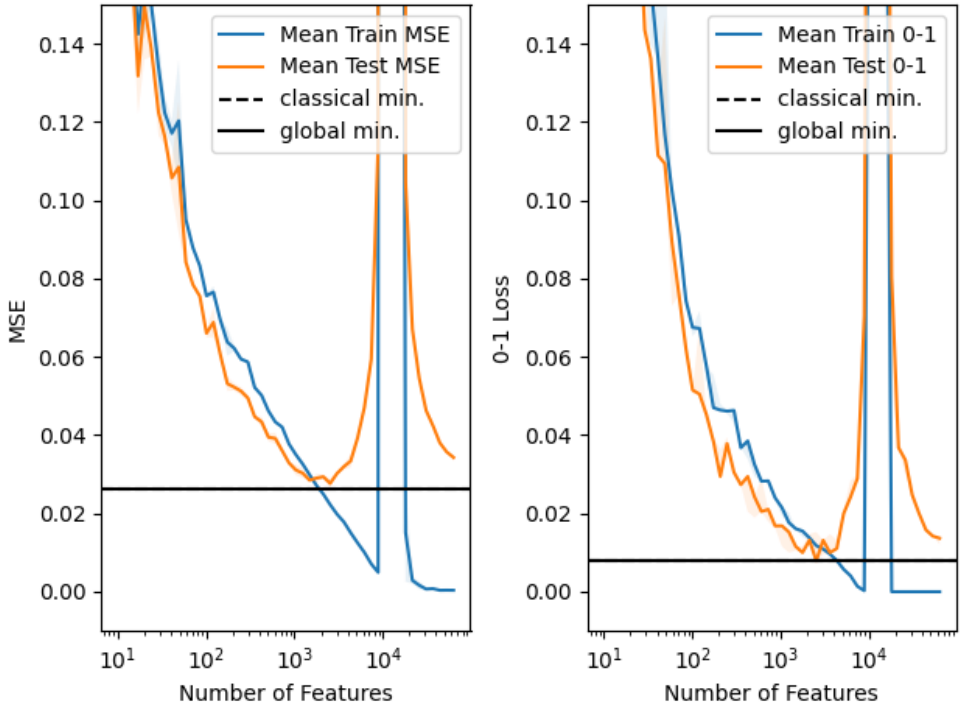
- First Experiment: Analyzing the performance of our random Fourier features model in the absence of noise. As shown in Figure 5.11, this model aligns with the double descent theory; there is a gap between the classical minimum and the global minimum of the test error. Here, the global minimum of the test loss is equal to about 0.02, while the classical minimum is strictly greater. The global minimum of the test error is strictly greater than 0.00 and the corresponding classical minimum almost overlaps. There are no visible standard deviations. This means that this model performs uniformly through different runs. The MSE losses show a strictly greater gap than the 0-1 losses. The model interpolates the training data at a level of complexity strictly lower than 10000 features. The model does not show benign, tempered or catastrophic overfitting; the test error shows a "double descent" regime.

Figure 5.11: Mean losses and mean errors in the absence of noise.



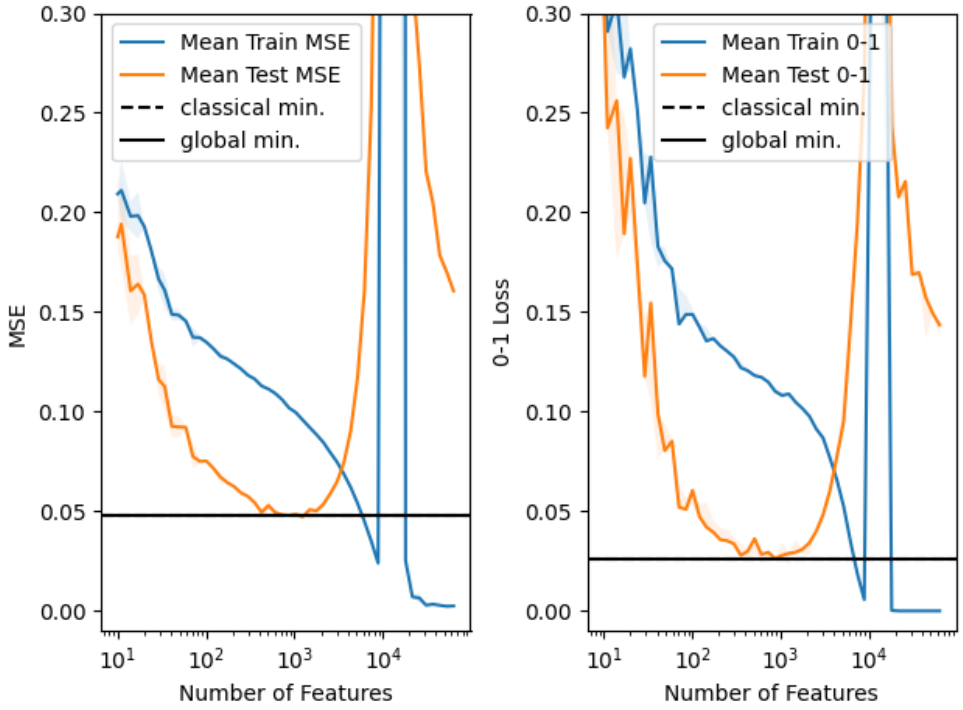

- Second Experiment: Analyzing the performance of our random Fourier features model in the presence of 1% noise in the training set. As shown in Figure 5.12, this model does not align with the double descent theory; the classical minimum and the global minimum of the test error overlap. Here, the global minimum of the test loss is strictly greater than 0.02, while the global minimum of the test error is similar to the previous experiment. There are no visible standard deviations. This means that this model performs uniformly through different runs. The MSE losses show a strictly greater gap than the 0-1 losses. As expected, introducing levels of noise equal to 1% in the training set leads to slowing down the descents of the errors and the losses and increasing the interpolation threshold and the gap between the losses. The gap between the errors is similar to the previous case. The model interpolates the training data at a level of complexity strictly greater than 10000 features. The model does not show benign, tempered or catastrophic overfitting; the test error shows a "double descent" regime.

Figure 5.12: Mean losses and mean errors with 1% noise in the training set.



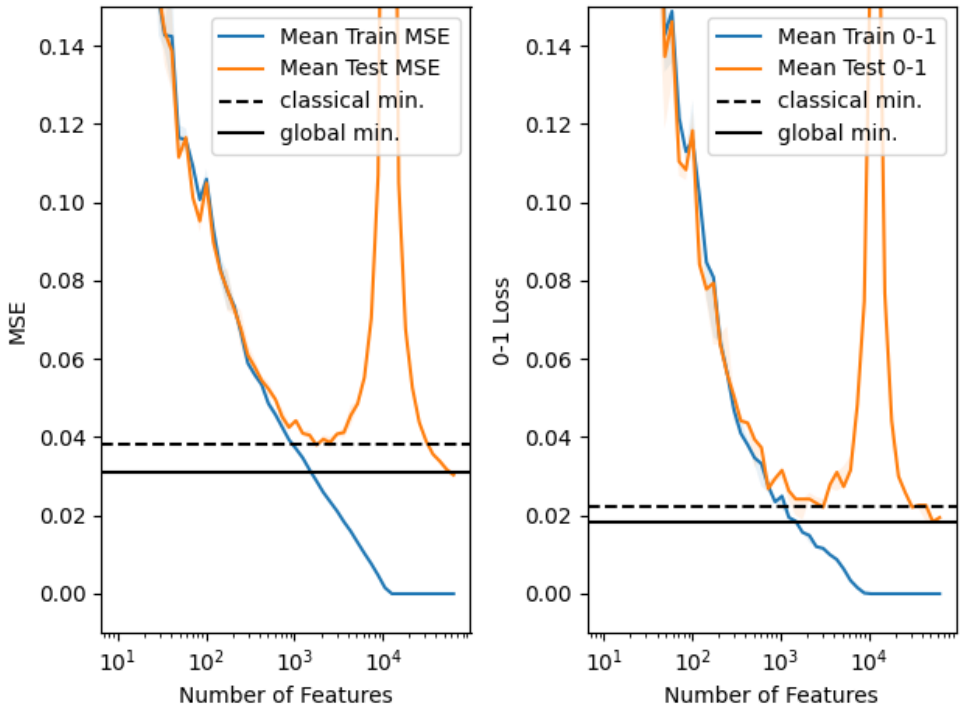

- Third Experiment: Analyzing the performance of our random Fourier features model in the presence of 10% noise in the training set. As shown in Figure 5.13, this model does not align with the double descent theory; there is no gap between the classical minimum and the global minimum of the test error. Here, the global minimum of the test loss is equal to about 0.05, while the global minimum of the test error is equal to about 0.25. There are no visible standard deviations. This means that this model performs uniformly through different runs. The MSE losses show a strictly greater gap than the 0-1 losses. As expected, introducing levels of noise equal to 10% in the training set leads to slowing down the descents of the errors and the losses and increasing the interpolation threshold and the gaps between the errors and the losses. The model interpolates the training data at a level of complexity greater than 10000 features. The model does not show benign, tempered or catastrophic overfitting; the test error shows a "double descent" regime.

Figure 5.13: Mean losses and mean errors with 10% noise in the training set.



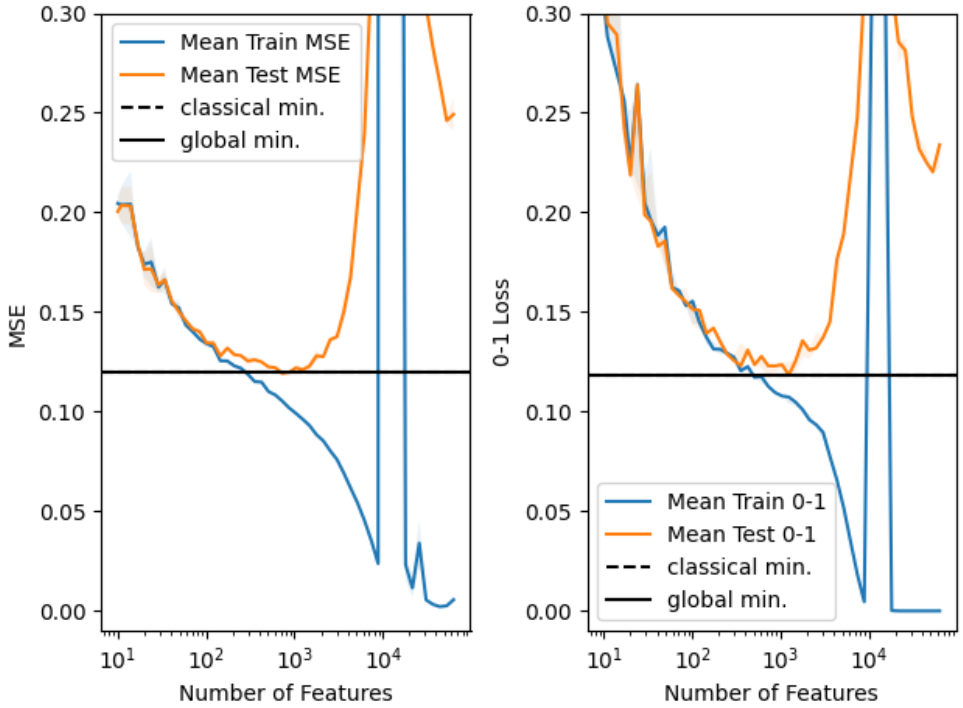

- Fourth Experiment: Analyzing the performance of our random Fourier features model in the presence of 1% noise both in the training set and test set. As shown in Figure 5.14, this model aligns with the double descent theory; there is a gap between the classical minimum and the global minimum of the test error. Here, the global minimum of the test loss is equal to about 0.03, while the classical minimum is equal to about 0.04. The global minimum of the test error is equal to about 0.02 and the corresponding classical minimum almost overlaps. There are no visible standard deviations. This means that this model performs uniformly through different runs. The MSE losses show a strictly greater gap than the 0-1 losses. As expected, compared with the first experiment, introducing levels of noise equal to 1% both in the training set and test set leads to slowing down the descents of the errors and the losses and increasing the gaps between the errors and the losses. The model interpolates the training data at a level of complexity similar to the case of the absence of noise. Compared with all other cases, the training error of this model decreases without rising. The model does not show benign, tempered or catastrophic overfitting; the test error shows a "double descent" regime.

Figure 5.14: Mean losses and mean errors with 1% noise both in the training set and test set.




- Fifth Experiment: Analyzing the performance of our random Fourier features model in the presence of 10% noise both in the training set and test set. As shown in Figure 5.15, this model does not align with the double descent theory; the classical minimum and the global minimum of the test error overlap. Here, the global minimums of the test loss and the test error are almost equal to about 0.12. There are no visible standard deviations. This means that this model performs uniformly through different runs. The MSE losses show a strictly greater gap than the 0-1 losses. As expected, compared with all the previous experiments, introducing levels of noise equal to 10% both in the training set and test set leads to slowing down the descents of the errors and the losses and increasing the gaps between the errors and the losses. The model interpolates the training data at a level of complexity similar to the third experiment. The model does not show benign, tempered or catastrophic overfitting; the test error shows a "double descent" regime.

Figure 5.15: Mean losses and mean errors with 10% noise both in the training set and test set.




Contrary to the theory, these experiments demonstrate that our random Fourier features models do not tend to align with the double descent theory. This means that the double descent phenomenon is not easily observable and very high levels of complexity are not sufficient to see it. Increasing the model complexity can lead to slightly better performance in terms of the test error. The presence of noise leads to a huge increment of the interpolation threshold, which needs basic significant levels of complexity. Increasing the levels of noise also implies slowing down the descents and increasing the gaps between the errors and the losses. The choice of the loss function has no significant impact on the performance of these models. In each model, the 0-1 losses and the MSE losses show similar trends in all the previous cases. The models do not show benign, tempered or catastrophic overfitting; the test errors show a "double descent" regime. Our models also always interpolate the training data and perform well on test data. High levels of noise are not sufficient to lead to a catastrophic overfitting regime, so these models are robust to the noise. These experiments confirm that most modern machine learning models, including random Fourier features models, tend to interpolate the training data and perform well on new data even in high amounts of noise and the choice of the loss function has no significant impact on the model performance. Contrary to the theory, these models do not tend to show tempered overfitting; the test error has a "double descent" trend.

# Chapter 6

# Conclusions

Our experiments have revealed both neural network models and random ReLU features models did not align with the double descent phenomenon, while the test error of random Fourier features models shows the double descent in some cases. Contrary to the theory, this suggests that the double descent phenomenon is not easily observable, and very high levels of complexity are not a sufficient factor to see it. We found that interpolation can be achieved with low levels of complexity for neural network models, even in the presence of significant levels of noise both in the training set and test set. These neural networks performed well also on test data, even in the presence of high amounts of noise. The increment of the interpolation threshold with noise is modest. Random features models showed some differences. Interpolation in these models required significant complexity, and the interpolation threshold saw a substantial increase with the presence of noise. The experiments also highlighted that random ReLU features models performed similarly both with the 0-1 loss and the binary cross-entropy loss in most cases. The same goes for random Fourier features models where they performed similarly both with the 0-1 loss and the MSE loss in most cases. So, the choice of the loss function has no significant impact on the performance of these models, aligning with the Belkin experiments. High levels of noise influence the training process slowing down the descents of the errors and the losses and increasing the gaps among them and the interpolation threshold, but they do not lead to catastrophic overfitting. This indicates that these models tend to be robust in the presence of noise. Our models mostly interpolate the training data and perform well on test data, even in the presence of high amounts of noise, showing benign overfitting in most cases. Given this, our experiment validates the Belkin contributions according to which most machine learning models tend to interpolate the training data and perform well on test data even in high levels of noise and the choice of the loss function has no significant impact on the performance.

# Bibliography


[BHMM19] Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116:15849–15854, 2019. doi:10.1073/pnas.1903070116.

[BMM18] Mikhail Belkin, Siyuan Ma, and Soumik Mandal. To understand deep learning we need to understand kernel learning. Technical report, UC San Diego, 2018. URL: https://arxiv.org/abs/1802.01396.

[CEL+23] Lorenzo Ciampiconi, Adam Elwood, Marco Leonardi, Ashraf Mohamed, and Alessandro Rozza. A survey and taxonomy of loss functions in machine learning. *CoRR*, 2023. doi:10.48550/ARXIV.2301.05579.

[DDPR15] Sumit Das, Aritra Dey, Akash Pal, and Nabamita Roy. Applications of artificial intelligence in machine learning: Review and prospect. *International Journal of Computer Applications*, 115:31–41, 2015. doi:10.5120/20182-2402.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[HM15] Mohammad Hossin and Sulaiman M.N. A review on evaluation metrics for data classification evaluations. *International Journal of Data Mining & Knowledge Management Process*, 5:01–11, 2015. doi:10.5121/ijdkp.2015.5201.

[JWH+23] Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani, and Jonathan Taylor. *An Introduction to Statistical Learning with Applications in Python*. Springer, 2023.

[KB14] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 2014. URL: https://arxiv.org/abs/1412.6980.

[KD20] Ku. Chanya A. Khanzode and Dr. Ravindra D.Sarode. Advantages and disadvantages of artificial intelligence and machine learning: A literature review. *International Journal of Library & Information Science*, 9(1):30–36, 2020. doi:10.17605/OSF.IO/GV5T4.

[Led21] Johannes Lederer. Activation functions in artificial neural networks: A systematic overview. Technical report, Department of Mathematics Ruhr-University Bochum, 2021. URL: https://arxiv.org/abs/2101.09957.

[MSA+22] Neil Mallinar, James B. Simon, Amirhesam Abedsoltan, Parthe Pandit, Mikhail Belkin, and Preetum Nakkiran. Benign, tempered, or catastrophic: A taxonomy of overfitting. Technical report, UC San Diego, 2022. URL: https://arxiv.org/abs/2207.06569.

[RR07] Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. volume 20, 2007. URL: https://dl.acm.org/doi/10.5555/2981562.2981710.

[Rud17] Sebastian Ruder. An overview of gradient descent optimization algorithms. Technical report, Insight Centre for Data Analytics, NUI Galway Aylien Ltd., Dublin, 2017. URL: https://arxiv.org/abs/1609.04747.

[SGT19] Yitong Sun, Anna Gilbert, and Ambuj Tewari. On the approximation properties of random relu features. Technical report, University of Michigan, 2019. URL: https://arxiv.org/abs/1810.04374.

[YS20] Li Yang and Abdallah Shami. On hyperparameter optimization of machine learning algorithms: Theory and practice. *Neurocomputing*, 415:295–316, 2020. doi:10.1016/j.neucom.2020.07.061.