Mathematical Machine Learning

Yahya Saleh, Tizian Wenzel, Kamal Sharma

March 26, 2024

Contents

Introduction

Underlying the success of artificial intelligence are learning algorithms, i.e., algorithms that learn from data to perform a certain task. We start by two concrete examples of supervised learning algorithms. In the first example we consider the problem of approximating functions from pointwise evaluations using linear regression. In the second example we look at the task of classifying hand-written digits. In these two examples we identify and familiarize ourselves with the main components of learning algorithms; *datasets*, *a hypothesis class*, and *optimization algorithms*. We further identify important aspects of supervised learning algorithms, such as overfitting, and underfitting. Finally, we motivate in these two examples problems at the forefront of research in mathematical machine learning, namely *the curse of dimensionality (CoD)* and *double/multiple descent phenomenon*.

I.1 Supervised Learning

I.1.1 Supervised Leaning Informally

In supervised learning tasks the dataset D is made up of two components, input variables $D_x = \{x_i\}_{i=1}^N$ and targets $D_y = \{y_i\}_{i=1}^N$. The dataset is assumed to be generated by an unknown function $f : \text{input} \to \text{target}$. The goal in a supervised learning task is to approximate the unknown function f pointwise, i.e., to find a function h such that $h(x) \approx f(x)$ for any x, whether it belongs to D_x or not. The target value can take finitely many values, e.g., target $\in \{0, 1, \dots, M\}$. In such a case, the supervised learning task is called a *classification task*. If the target can take infinitely many values, the learning task is called a *regression task*. Supervised learning problems are approached by first choosing a *hypothesis space* $\mathfrak{H}$, in which one looks for an approximation to the unknown function f. For example, if the data x is one-dimensional and the target take values in $\mathbb{R}$ one can define the hypothesis class to be the set of all affine mappings, i.e.,

$$\mathfrak{H} = \big\{ f \mid f(x) = ax + b,\ a, b, \in \mathbb{R} \big\}. \tag{I.1}$$

Then, one define a loss function l that measures how well a hypothesis function h approximates an unknown function f at a point x. Using the dataset D, the supervised learning problem can be then formulated as an optimization problem

$$\min_{h \in \mathfrak{H}} \frac{1}{N} \sum_{i=1}^{N} l(h(x_i), y_i). \tag{I.2}$$

While there are many alternatives to solve these optimization problems, the by-far most used algorithms are variants of the gradient-descent algorithm.

Let's look at some concrete examples.

Example I.1. [Regression] ☛ I.1 Let x be a random variable that takes values in the interval $[-1, 1]$. And assume we have access to a dataset $D = \{(x_i, y_i)_{i=1}^{200}\}$ generated by the unknown function

$$f(x) = x^2 \cos(5x) \exp(-x).$$

Assume that the dataset is corrupted by Gaussian noise. To learn a function h that approximates f let your hypothesis class be the class of affine functions (I.1). Let the loss function be the absolute error, i.e.,

$$\begin{aligned} l(h(x_i), y_i) &= |h(x_i) - y_i| \\ &= |ax_i + b - y_i|. \end{aligned}$$

Use a gradient-descent-like algorithm to choose the best hypothesis h, i.e., the best scalars a and b.

Change your hypothesis class to the class of all polynomials up to order 20 and repeat the optimization process. Which class is better for optimization? Figure I.1 shows the outcome of such an experiment.

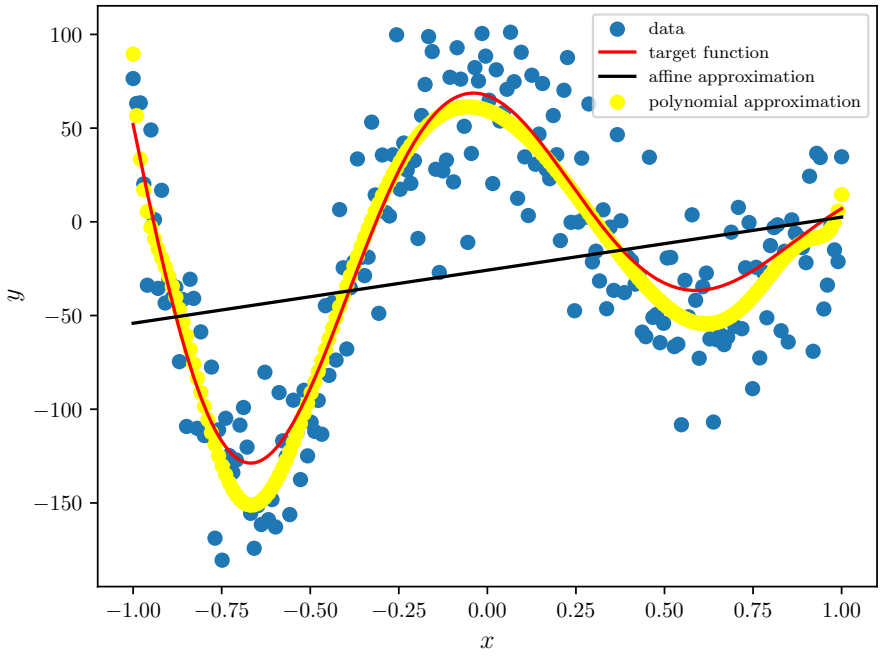

Figure I.1: A regression task; the goal is to fit noisy data (blue dots) assumed to be generated from a true function (solid red line). The data is fitted using an affine mapping (solid black line) and a polynomial mapping (solid yellow line).

Example I.2. [Classification] 👉 I.2 We consider a classification problem of hand-written digits. The input to the problem is an 8×8 image of a hand-written digit, and the output should be the predicted value of the digit. Formally, we consider x to be a random variable taking values in $[0, 16]^{8\times 8} \subset \mathbf{N}^{8\times 8}$, i.e., x is a random variables in a matrix representation, where each matrix element takes an integer value between 0 and 16. Here, the value of a certain matrix element represents its color, where 0 denotes black, and 16 denotes white. Let the target value y be a random variable taking values in the discrete set $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$. To solve this supervised learning problem we consider as a hypothesis class the following multilayer perceptron:

$$\mathfrak{H} = \left\{\text{softmax } w_2 \left(\sigma(w_1 \cdot x + b_1)\right) + b_2; w_1 \in \mathbb{R}^{\text{dh},8}, b_1 \in \mathbb{R}^{\text{dh}}, w_2 \in \mathbb{R}^{10,\text{dh}}, b_2 \in \mathbb{R}^{10}\right\},$$

where dh is called the number of hidden units. In this class, the linear parameters are the weight matrices w_1, w_2 and the biases b_1, b_2. σ is a nonlinear non-learnable function, often referred to by the *activation function*. A common choice is the ReLU (Rectified Linear Unit) function

$$\text{ReLU}(x) = \max(0, x)$$

The softmax function (or layer) takes a set of real-valued input and transforms it into a probability distribution over multiple classes. In our example we have ten classes and the softmax is given by

$$\text{softmax}(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{10} e^{z_j}}.$$

Therefore, the output of the hypothesis function is a probability distribution over the 10 classes. Concretely, the output is 10-dimensional, where each entry denotes the probability of the input image to represent a certain digit.

For facilitating the implementation we represent the target value as a one-hot vector. For example, given a target value 4, we represent it as the vector $y = (0, 0, 0, 1, 0, 0, 0, 0, 0, 0)$. A suitable loss function for such problems is the categorical cross entropy function given by

$$l(h(x_i), y_i) = \sum_{c=0}^{9} y_i^c \log(h(x_i)^c),$$

where (x_i, y_i) is a specific training example. y_i^c refers to the c-th entry of the one-hot vector representation of the target.

Compute the training and test errors and study how they change when changing the number hidden units or number of layers.

Figure I.1 shows two hypotheses, one linear and one nonlinear, that we learned to fit the data in Example I.1. The figure depicts an interesting phenomenon; a certain hypothesis h can fit the data too accurately; notice for example in Figure I.1 that the polynomial-regression model fits badly local minima of the target function. In these regions, it is optimized to fit the noise. The outcome of such a result is that the polynomial-regression model will fail to generalize well in these regions, i.e., it will have large error on unseen data in these regions. This is called *overfitting*. On the other hand, the linear-regression model produces largely deviated from the data everywhere, and would, hence, also generalizes badly to unseen data. This is called an *underfitting* phenomenon.

Think about the influence of the following factor on the underfitting and overfitting:

- Complexity of the model. For a polynomial-regression model this can be the degree of the polynomial.
- Size of the dataset. For example, would adding more data decrease or increase underfitting?

I.1.2 Supervised Learning Formally

A formal definition of learning.

I.2 This Course

What we will cover at what we want.

I.2.1 Curse of Dimensionality

Programming exercise and theorem about approximating smooth functions in higher dimensions.

Example I.3. [Classification] 👉 I.3 tba.

I.2.2 Approximating Highly-Oscillatory Functions

Example I.4. [Classification] 👉 I.4 tba.

I.2.3 Validity of Ocaam Razor and Double-Descent Phenomenon

Example I.5. [Double descent] 👉 I.5 tba.

Wait! What is what?

Here is a list of questions that help you check your understanding of key concepts inside this chapter?

Empirical Risk Minimization Principles

Machine Learning Models

III.1 Neural Networks

III.2 Kernel Methods

Modern Machine Learning

Modern Mathematical Machine Learning