Mathematical Introduction to Machine Learning

Lecture 1: An Overview of Machine Learning

May 9, 2025

Lecturer: Lei Wu *Scribe: Lei Wu*

In this lecture, we provide an overview of machine learning (ML), specifically focusing on supervised learning from a mathematical viewpoint. Special emphasis is placed on the challenges and opportunities that arise when dealing with high-dimensional data.

1 Introduction

In supervised learning, we are given n samples $(x_1, y_1), \cdots, (x_n, y_n)$ with

$$y_i = f^*(x_i) + \varepsilon_i,$$

where:

- x_i's are the inputs and let $x_i \in \mathcal{X}$, where $\mathcal{X}$ represents the input domain.
- y_i's are the labels and let $y_i \in \mathcal{Y}$, where $\mathcal{Y}$ represents the output/label domain.
- ε_i's represent the noise.
- $f^* : \mathcal{X} \mapsto \mathcal{Y}$ denotes the target function, also known as the ground truth, the label function, or the regression function.

Note that we denote by $\mathcal{X}, \mathcal{Y}$ the input and output domains, respectively. The collection $S = \{(x_i, y_i)\}_{i=1}^n$ is referred to as the training (data)set, as we use these data to train our machine learning models.

Remark 1.1. In theoretical analysis, it is often assumed that $x_1, \ldots, x_n$ are i.i.d. (Independent and Identically Distributed) samples drawn from an input distribution ρ. It should be noted that this i.i.d. assumption may not always hold in practice.

The aim of supervised learning is to approximate/recover/learn the unknown target function f^* by using the training samples S along with some potential prior knowledge about f^* (e.g., f^* might be known to be rotationally or permutation invariant). Roughly, supervised learning problems can be categorized as follows:

- **Regression:** The labels take real or continuous values; for example, $\mathcal{Y} = \mathbb{R}$. Regression problems are commonly found in applications, such as financial forecasting, risk assessment, and scientific computing. Figure 1 illustrates an example of learning a univariate function in a regression setting.
- **Classification**: The labels are discrete, for example $\mathcal{Y} = \{1, 2, 3, \ldots, 10\}$. Note that the numerical ordering of the labels is not inherent but imposed for representation. Classification problems include applications, such as image recognition and semantic analysis.

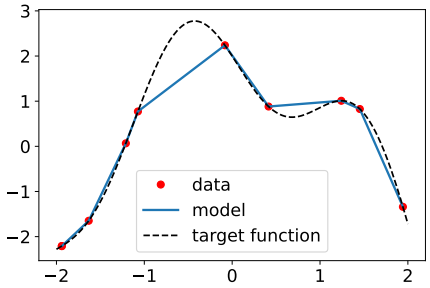


Figure 1: Learning a univariate function using only finite samples. Here, the model is a piecewise linear function.

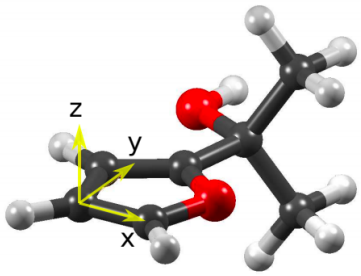
Low dimension vs. high dimension. It is worth noting that the univariate function regression problem shown in Figure 1 is low-dimensional and can be effectively addressed using traditional methods like (piecewise) polynomial interpolation.

One major advantage of modern ML over traditional methods is its ability to handle high-dimensional problems efficiently. Below are examples of such high-dimensional problems where ML has proven to be invaluable:

- **Image recognition.** Shown below is the task of ImageNet. This dataset has around 1 million $224 \times 224 \times 3$ color images as well as the labels. The labels are image categories, e.g. bird, cat, and the number of categories is 1000. The aim is to learn a classifier: $[0,1]^{224\times224\times3} \to \{1, 2, \ldots, 1000\}$. Here, the input dimension is $d = 150528 \gg 1$.

Figure 2: The ImageNet website: https://www.image-net.org/update-mar-11-2021.php

- **Molecule modeling.** Learn the potential energy function f^*, which maps the atom coordinates to the potential energy of that molecule. Assume that there are N atoms. Then, we know that $f^* : \mathbb{R}^{3\times N} \mapsto \mathbb{R}$ must satisfy the following symmetries/invariances:
  - Translation and rotation,
  - Permutation among identical atoms.



- **Image translation.** Learn a map $\mathcal{T} : [0,1]^{214\times214\times3} \mapsto [0,1]^{214\times214\times3}$. In such a case, the output space $\mathcal{Y}$ is high-dimensional too.

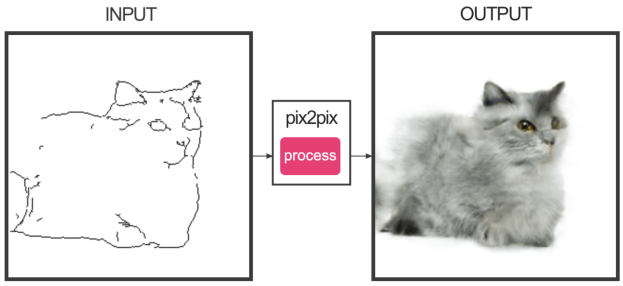


Figure 3: https://affinelayer.com/pixsrv/

- **Solving PDEs.** Learn the solution map: $u = S(f)$, which maps the source term f directly to the solution u. In this case, both the input f and output u are functions, and thus both the input and output spaces are **infinite** dimensional:

$$\mathcal{L}(u, \nabla u, \nabla^2 u) = f. \tag{1}$$

Remark 1.2. For all the examples above, the input space is **high-dimensional**, and the target functions are complex nonlinear functions. In some cases, even the output space is also high-dimensional or infinite-dimensional. These challenges often require us to use large models $f(\cdot;\theta)$ capable of approximating the target functions effectively. This is where machine learning models, particularly neural networks, show their strength in handling high-dimensional data.

1.1 Learning paradigm

The most popular way of doing machine learning can be decomposed into three steps.

- **Step 1:** Choose an appropriate (parametric) model $f(\cdot;\theta)$ with $\theta \in \mathbb{R}^m$. Here, θ denotes the parameters to be learned from data and m denotes the parameter/model size. Then $\mathcal{F}_m = \{f(\cdot;\theta) : \theta \in \mathbb{R}^m\}$ forms our *hypothesis space/class* (also called the model space/class). The commonly used models include
  - linear functions: $f(x;\theta) = \theta^T x + \theta_0$,
  - Fixed basis expansions: $f(x;\theta) = \sum_{j=1}^m \theta_j \varphi_j(x)$, where $\{\varphi_j\}$ are a set of basis. Typical bases include polynomials, trigonometric functions, wavelets, random features, piecewise polynomials (aka splines).
  - (Deep) neural networks.
- **Step 2:** Formulate an optimization problem. The task of "learning" is to select a $f \in \mathcal{F}_m$ such that f is close to f^* by using the training data S. This can be formulated as to minimize the empirical risk

$$\min_\theta \widehat{\mathcal{R}}(\theta) := \frac{1}{n}\sum_{i=1}^n \ell(f(x_i;\theta), y_i),$$

  where $\ell : \mathcal{Y} \times \mathcal{Y} \mapsto [0, \infty)$ is a loss function. The empirical risk measures the fitting error at the training data.

Sometimes, we may have some prior information about f^* (e.g., f^* is smooth). To use this information, we can design a penalty function $p(\cdot)$ to encode this prior information. As a result, we will minimize the following regularized risk

$$\min_{\theta} \widehat{\mathcal{R}}(\theta) + \lambda\, p(\theta). \tag{2}$$

The choice of $p(\cdot)$ is to ensure that when the value of $p(\theta)$ is small, the corresponding model $f(\cdot;\theta)$ is close to f^*. Here, λ is called the hyperparameter/tuning parameter, which balances the fitting error and penality. *Different from the model parameters θ, the hyperparameter is not learned from data.*

- **Step 3:** Choose an optimization method to solve the problem (2). Common methods include gradient descent, stochastic gradient descent, and various second-order methods like Newton's method.

Measure the model performance. Denote by $f(\cdot;\hat{\theta}_n)$ the model generated by the above learning procedure. Then the next question is: How good is the the learned model $f(\cdot;\hat{\theta}_n)$? This is measured by using the generalization error/excess risk:

$$\mathcal{E}(\theta) = \mathbb{E}_{x\sim\rho}[\ell(f(x;\theta), f^*(x))], \tag{3}$$

which quantifies how the learned model generalizes to unseen data points. Keep in mind that our objective is to minimize the generalization error instead of the empirical risk. However, we can only deal with the empirical risk since only training data are available.

In practice, it is impossible to compute the population risk (3) exactly. We often approximate the population risk by Monte-Carlo integration over a test set: S_{test}, the resulting error is called the *test error*. In comparison, the empirical risk is often called the *training error*.

2 Dissecting generalization errors

Here we attempt to understand how the generalization error depends on the parameter size m (i.e., the model complexity) and sample size n. Does increasing the parameter size/model complexity always reduce the generalization error of the learned model? To this end, we introduce the following the solution

$$\theta^* = \operatorname*{argmin}_{\theta} \mathcal{R}(\theta). \tag{4}$$

Note that $f(\cdot;\theta^*)$ denote the best solution in the hypothesis space $\mathcal{F}_m$ for approximating f^*.

Let $f_\theta = f(\cdot;\theta)$. Consider the following decomposition of the generalization error:

$$\underbrace{f_{\hat{\theta}} - f^*}_{\text{gen-err}} = \underbrace{f_{\hat{\theta}} - f_{\theta^*}}_{\text{estimation error}} + \underbrace{f_{\theta^*} - f^*}_{\text{approximation error}}. \tag{5}$$

Let us discuss the two terms of the RHS separately.

- **Approximation error.** This error is independent of the sample size and only depends on the hypothesis space. In general, the larger is the hypothesis space, the smaller is the approximation error. If $f^* \in \mathcal{F}_m$, this error becomes zero.
- **Estimation error.** This error is caused by the fact that we have only finite training samples. We are unable to identify the best solution θ^* with only finite data.

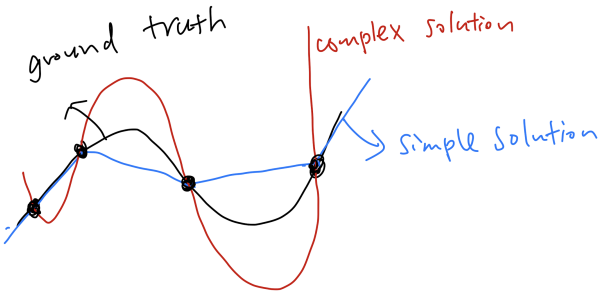


Figure 4: An illustration of the estimation error. The black dots represent the training data. The black curve is the ground truth. The blue and red curves denote two solutions in the hypothesis space. For larger models, it is easier to select bad solutions (the red curve).

- **Selection uncertainty.** We first assume that the data are clean, i.e., $y_i = f^*(x_i)$. Figure 4 provides an illustration of the estimation error in this case. Note that there are many solutions f's in $\mathcal{F}_m$ such that they all perfectly fit the n data points; some of them may generalize very badly. This causes uncertainty in selecting solutions by using only finite data points and inevitably a bad solution is selected. The larger the hypothesis space/model is, the worse those bad solutions are and the easier the bad solution is selected.
  - Note the regularization can alleviate the issue. But there is no perfect regularization that can resolve it.
  - The more data we have, the smaller is the uncertainty, thereby the generalization error.
- **Overfitting?** In most classic textbooks, the estimation error is explained by using the phenomenon of "overfitting". When there are noises in data, complex models are easier to fit the noise than simple models. This overfitting of noises inevitably causes a type of error. Nevertheless, as explained above, the estimation error exists and increases with the model complexity even when the data are completely clean. In such a case, "overfitting" is a bad explanation of the estimation error, since there is no noise to overfit.

2.1 The generalization error curve

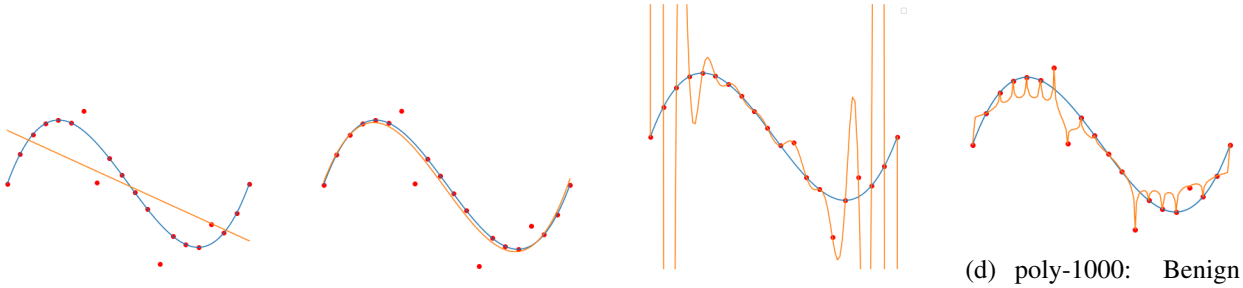
Here we attempt to understand how the generalization error changes with the model size. Let n, m denote the sample size and number of parameters, respectively. Let $g(n, m), e(n, m), a(m)$ denote the generalization error, estimation error, and approximation error, respectively. Roughly, we have

$$g(n, m) = a(m) + e(n, m).$$

In general, we only know that

- $a(m)$ is decreasing with m.
- $e(n, m)$ is decreasing with n and increasing with m [1];

[1] The assumption that $e(n, m)$ increases with m is not always true. When increasing m, the model class always becomes larger. However, the subset that ML methods can explore may become smaller in some cases. Indeed, there exist methods, which provide simpler models when m is larger. We will see it in the future.



(a) poly-1: Under fitting (b) poly-3: Nice fitting (c) poly-20: Overfitting (d) poly-1000: Benign overfitting

Figure 5: The trade-off between estimation error and approximation error.

Hence, there must exist trade-offs between the approximation error and estimation error for obtaining the optimal generalization error. We ask the question:

For fixed n, how $g(n, m)$ behaves when increasing m?

- The most common picture is the U-shaped curve shown in the left subplot of Figure 6, suggesting that the generalization error first decreases then increases.
- **Is the U-shaped picture really true?** In general, we only know that $e(n, m)$ decreases and $a(m)$ increases with m. The middle and right subplots in Figure 6 show that we can construct $e(n, \cdot)$ and $a(\cdot)$ such that $g(n, \cdot)$ exhibits very complicated multiple-descent behaviors. These curves are artificially constructed but they indeed reveal the potential complexity of the generalization error curve. Do these curves really exist in practical models? The answer is affirmative, e.g., the random feature models and neural networks [Belkin et al., 2019].

In a word, the shape of the generalization error curve depends on the relative speed between approximation error and estimation error. For simple tasks (e.g., the linear regression), the approximation error goes to zero extremely fast, yielding a simple U-shaped curve. However, for learning high-dimensional complex nonlinear functions, the approximation error cannot be ignored even when m is very large. In such a case, the generalization error curve may exhibit complex behaviors.

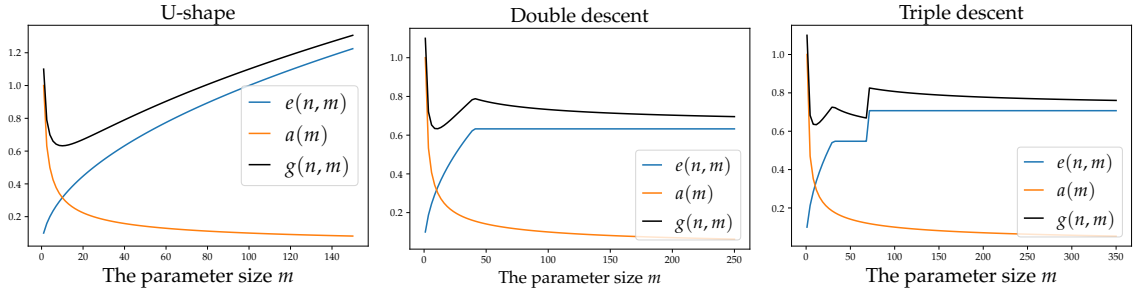


Figure 6: Generalization error vs. the model size m. **Left:** A U-shaped curve; **Middle:** A double descent curve; **Right:** A triple descent curve.

3 Understanding the curse of dimensionality

We have claimed many times that learning high-dimensional nonlinear functions are extremely hard. Next, we provide both intuitive explanations and concrete quantification of this hardness.

3.1 A simple example

In this section, we consider piecewise constant models. To simplify notations, we denote by by 1_Ω, for a $\Omega \subset \mathbb{R}^d$, the indicator function:

$$1_\Omega(x) = \begin{cases} 1 & \text{if } x \in \Omega \\ 0 & \text{if } x \notin \Omega. \end{cases}$$

For a target function f^*, consider the piecewise constant interpolation:

$$\hat{f}_m(x) = \sum_{j=1}^{m} f^*(x_j) 1_{S_j}(x),$$

where $S_j = [x_{j-1}, x_j]$. Here $x_j = \frac{j}{m} = jh$ with $j = 1, \ldots, m$. Here $h = \frac{1}{m}$ is the grid size.

Lemma 3.1. *For any $f \in C^1([0,1])$, define $\mathrm{Lip}(f) = \sup_{x\in[0,1]} |f'(x)|$. Then,*

$$\|\hat{f}_m - f^*\|_{L^2} \le \frac{C \,\mathrm{Lip}(f^*)}{m}.$$

Proof. Using the Lipschitz property, we have

$$\begin{aligned} \|\hat{f}_m - f^*\|_{L^2}^2 &= \sum_{j=1}^{m} \int_{x_{j-1}}^{x_j} |f^*(x_{j-1}) - f^*(x)|^2 \,\mathrm{d}x \le \sum_{j=1}^{m} \int_{x_{j-1}}^{x_j} \mathrm{Lip}^2(f^*) |x_{j-1} - x|^2 \,\mathrm{d}x \\ &\le \mathrm{Lip}^2(f^*) m \frac{h^3}{3} = \mathrm{Lip}^2(f^*) \frac{h^2}{3} = \frac{\mathrm{Lip}^2(f^*)}{3m^2}. \end{aligned}$$

□

Now let us proceed to a high-dimensional case. Consider the domain $[0,1]^d$, for which S_j is a cube of volume h^d and we have $m = h^{-d}$ cubes.

Theorem 3.2. *For any $f \in C^1([0,1]^d)$, define $\mathrm{Lip}(f) = \sup_{x\in[0,1]^d} \|\nabla f(x)\|_2$. Then,*

$$\|\hat{f}_m - f^*\|_{L^2} \le \frac{C\sqrt{d}\,\mathrm{Lip}(f^*)}{m^{1/d}},$$

where C is an absolute constant.

Proof. By definition,

$$\begin{aligned} \|\hat{f}_m - f^*\|_{L^2}^2 &= \sum_{j=1}^{m} \int_{S_j} |f^*(x_{j-1}) - f^*(x)|^2 \,\mathrm{d}x = \sum_{j=1}^{n} \int_{[0,h]^d} |f^*(x_{j-1}) - f^*(x_{j-1} + t)|^2 \,\mathrm{d}t \\ &\le \mathrm{Lip}^2(f^*) \sum_{j=1}^{m} \int_{[0,h]^d} \|t\|_2^2 \,\mathrm{d}t = \mathrm{Lip}^2(f^*) \sum_{j=1}^{m} d h^{d-1} \int_0^h t_1^2 \,\mathrm{d}t_1 \\ &= \frac{d\,\mathrm{Lip}^2(f^*)}{3} m h^{d+2} = \frac{d\,\mathrm{Lip}^2(f^*)}{3} h^2 = \frac{d\,\mathrm{Lip}^2(f^*)}{3m^{2/d}}. \end{aligned}$$

□

Curse of dimensionality (CoD). Let us look at the error rate $O(m^{-1/d})$. To reach an accuracy ε, we need $m = \Omega((1/\varepsilon)^d)$ grid points, which depends on the input dimension d exponentially. For instance, if would like an accuracy 0.1, we need $\Omega(10^d)$ grid points, which is impossible for large d. We often call this phenomenon the curse of dimensionality (CoD).

What causes the CoD?

- *What is the property/information of the target function that allows our model to generalize to unseen data points (outsize the grid points)?*

  The answer is the smoothness of f^*, i.e., $\mathrm{Lip}(f^*) < \infty$. This can be seen clearly from the proof, which depends on the local smoothness. The error in fact goes like $O(h)$ for all dimensions, where h quantifies the (average) distance between a test point x with the nearest point in the training set $\{x_i\}_{i=1}^n$.

- *Why is there a CoD when relying on the smoothness for generalization?*

  The error of the smoothness-based generalization is $O(h)$, proportional to the typical size among points. However, in a d-dimensional unit cube, we must need $m = h^{-d}$ points to reach this average distance. Thus, the number of points grows with d exponentially fast, causing the CoD.

3.2 Can we overcome the CoD?

How can we overcome the CoD? We must exploit some particular properties (beyond the traditional smoothness) of f^* to learn a model for generalization. But one needs to be careful about the choice of model $f(\cdot;\theta)$. If $f(\cdot;\theta)$ is not properly adaptive to the particular property of f^*, we may still suffer from CoD even if f^* is simple.

- If f^* is a general Lipschitz function, learning it suffers from CoD no matter what model is used.
- If f^* is a linear function, learning it with a piecewise-constant model still suffers from CoD.
- If f^* is a constant function, learning it with a piecewise-constant model does not suffer from CoD.
- If f^* is a linear function, learning it with linear regression does not suffer from CoD.

References

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