Mathematics of Deep Learning
Lecture 1: Introduction to empirical risk minimisation

Bruno Loureiro
Département d'Informatique, École Normale Supérieure - PSL & CNRS, France

10/01/2025

Typos, comments or suggestions? Get in touch at: bruno.loureiro@di.ens.fr

Introduction & Motivation

AI-based technology is increasingly present in day-a-day life. Tasks such as talking to your phone or asking a LLM to translate a text were unimaginable a decade ago. The backbone of these developments is *machine learning*, the field concerned with how computers learn from data. In particular, one class of machine learning models, *deep neural networks*, has been a driving force behind these developments.

Despite the major engineering breakthroughs achieved by neural networks, it is fair to say our current mathematical understanding of their working remains, to say the least, poor. Indeed, current machine learning practice defies several aspects of the intuition built from classical statistical analysis, presenting us with several interesting mathematical challenges. While this “gap with the practice” is frustrating to some, the mathematically minded student should see these challenges as a fertile ground with plenty of opportunity. Open problems are the fuel of research.

Curiously, an important part of the progress made in the past few years came from the understanding that the “exotic” properties of neural networks observed by practitioners are shared by simpler architectures that are amenable to a mathematical analysis. This observation will be the guiding thread of these lectures. Our goal will be to motivate and investigate these research questions in the simplest context where we can make sense of them.

⚠ A word of caution: as any research-level course, there is no consensus of what “Mathematics of deep learning” actually means. Therefore, it should go without saying that this course is an obviously biased selection of topics. It is not intended to be exhaustive, and probably a more appropriate name for this course would be “Some mathematics of deep learning”.

1 Supervised learning

Machine Learning is the subject concerned with the problem of how statistical models learn patterns from data. In practice, this can mean different things depending on the nature of the data and the nature of the information one aims to extract from it. Three of the most common tasks in machine learning are *supervised learning*, *unsupervised learning* and *reinforcement learning*. In the following, we will be focusing in supervised learning.

In supervised learning, data is given in pairs $\mathcal{D} = \{(x_i, y_i) \in \mathcal{X} \times \mathcal{Y} : i \in [n]\}$ (often refereed to as the *training data*), where $x_i \in \mathcal{X}$ are known as the *covariates* or *input data* and $y_i \in \mathcal{Y}$ are known as the *response* or *labels*. In particular, when $\mathcal{Y} = \mathbb{R}$, we say we have a *regression task*, while if $\mathcal{Y}$ is a discrete set, e.g. $\mathcal{Y} = \{1, \cdots, K\}$ we say we have a *classification task*. The main goal of supervised learning is to come up with a function $f : \mathcal{X} \to \mathcal{Y}$ that given a covariate, predicts its label “as well as possible”. A few comments are in order.

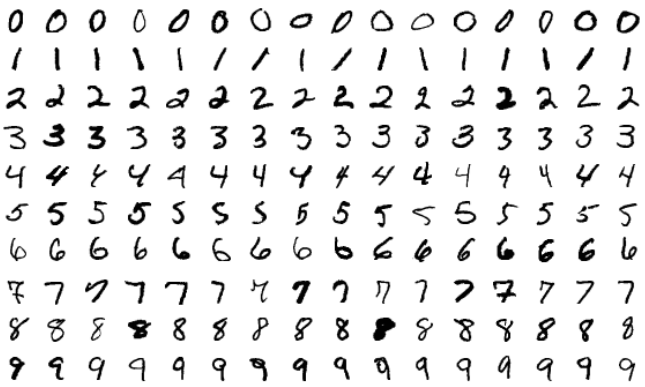

Figure 1: Example of different samples $x_i \in \mathbb{R}^{28\times 28}$ from the MNIST data set.

- Often, we will assume the covariates are vectorised $\mathcal{X} \subset \mathbb{R}^d$. In "real life", data is encoded in a data structure, for example an Excel table, a `pandas.DataFrame` or an image given by a `numpy.ndarray` with pixels and RGB values). Nevertheless, it is common practice to *vectorise* or *flatten* in a big vector. Studying how the encoding might affect prediction is an interesting topic, although outside of the scope of these lectures.

- In a K-class classification task, the image of the predictor $f : \mathcal{X} \to [K]$ is a discrete set. This is not amenable to doing optimisation (for instance defining a derivative). Therefore, in practice it is common to decompose $f = d \circ \boldsymbol{g}$ with $\boldsymbol{g} : \mathcal{X} \to \mathbb{R}^K$, known as the *score*, and $d : \mathbb{R}^K \to [k]$, known as the *decoder*.

- When $\mathcal{X} \subset \mathbb{R}^d$ and $\mathcal{Y} \subset \mathbb{R}$, it is common to define the *covariate matrix* $\boldsymbol{X} \in \mathbb{R}^{n\times d}$ and the *response vector* $\boldsymbol{y} \in \mathbb{R}$ by stacking the covariates x_i and the responses y_i row-wise.

Example 1 (Supervised learning tasks)**.** A few practical examples of supervised learning are:

- **Image classification:** The covariates x_i represent images and the labels y_i encode the respective classes. A very popular example is the MNIST data set, composed of $n = 70000$ images of black and white hand written digits. In this case, the covariates are given in terms of a matrix $\boldsymbol{x}_i \in \mathbb{R}^{28\times 28}$ with entries containing the grayscale level of each pixels, and $y_i \in \{0, \dots, 9\}$ label the respective digit. See Figure 1 for an illustration.

- **Price estimation:** Consider the problem of estimating the value of a house. In this case, the covariates can be a table with different relevant information (surface, number of rooms, location, electric efficiency, date of construction, etc.) and $y_i \in \mathbb{R}$ its price.

- **Text generation:** In this task, we are given a sentence (i.e. a sequence of words) and the goal is to predict the next word ("text completion"). In this case, the covariates are given by a vector of words, and the responses are the next word in the sentence. This task is at the heart of *Large Language Models* (LLMs) such as GPT, and is also known as *self-supervised learning* since the supervised data is typically obtained by taking full sentences and masking words.

The key word in the definition given above is "to learn as well as possible". There are two subtle aspects in this sentence: what does "learning" and "as well as possible" means.

Learning vs. memorisation — To see this, consider the following predictor:

$$f(x) = \begin{cases} y_i & \text{if } x = x_i \text{ for some } (x_i, y_i) \in \mathcal{D} \\ 0 & \text{otherwise} \end{cases} \tag{1.1}$$

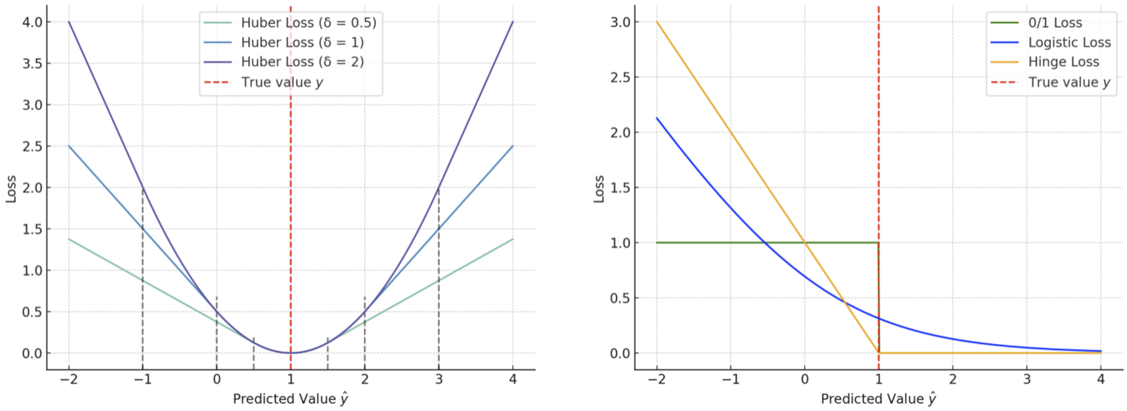

Figure 2: **(Left)** Huber loss. **(Right)** Classification losses.

It is clear that this function perfectly assigns the correct response to all covariates which are in the training set. However, it will very likely predict the wrong response for covariates which are not in the training data. Indeed, any reasonable definition of "learning" will imply that this predictor has not learned anything from the data, but rather *memorised* the training data.

To make the notion of "learning" precise, it is common to adopt a probabilistic point of view. More precisely, we assume that the training data $(x_i, y_i)_{i\in[n]}$ has been independently[1] drawn from a joint probability distribution $p(x, y)$ on $\mathcal{X} \times \mathcal{Y}$. With this assumption, we say that a predictor $f : \mathcal{X} \to \mathcal{Y}$ has *learned* is it is able to "predict well" the response of covariates which are outside the training data, i.e. that for a typical new sample $(x_{\text{new}}, y_{\text{new}}) \sim p$, $f(x_{\text{new}})$ is "close" to y_{new}. Now, it remains making precise what "close" means.

1.1 Loss and risk: or how to measure "good" learning

To quantify how "good" a predictor is, we introduce a notion of "distance"[2] between the predicted response $f(x)$ and the true response y, known as a *loss function* $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+$. From that, it is natural to define the following two notions:

Definition 1 (Risks)**.** Let $(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$ denote training data drawn from p, and let $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+$ denote a loss function. For any predictor $f : \mathcal{X} \to \mathcal{Y}$ we define the *empirical risk*, also known as the *training error*:

$$\hat{R}(f; \mathcal{D}) \coloneqq \frac{1}{n} \sum_{i=1}^{n} \ell(y_i, f(x_i)). \tag{1.2}$$

Similarly, we define the *population risk*, also known as *test* or *generalisation error*:

$$R(f) \coloneqq \mathbb{E}\left[\ell(y, f(x))\right]. \tag{1.3}$$

⚠ Note that while $\hat{R}(f; \mathcal{D})$ is a random function of f (since x_i, y_i are random variables), $R(f)$ is a deterministic function of f.

Example 2 (Loss functions for regression)**.** Consider a regression task $\mathcal{Y} = \mathbb{R}$ with vectorised data $\mathcal{X} = \mathbb{R}^d$.

[1] Although sample-wise independence is a common assumption, it is good to keep in mind it is not always the case in practice, where there might be sampling biases.

[2] Note the loss function is not a distance in the mathematical sense.

- **Square loss:** The widely used loss function for regression problems is the square loss:

$$\ell(y, f(\boldsymbol{x})) = \frac{1}{2}(y - f(\boldsymbol{x}))^2 \tag{1.4}$$

- **Absolute deviation:** The square loss is particularly sensitive to outliers in the data. An alternative loss function for regression which is less sensitive to outliers is the absolute deviation loss:

$$\ell(y, f(\boldsymbol{x})) = |y - f(\boldsymbol{x})| \tag{1.5}$$

- **Huber loss:** The Huber loss combines the squared and absolute deviation losses:

$$\ell_\delta(y, f(\boldsymbol{x})) = \begin{cases} \frac{1}{2}(y - f(\boldsymbol{x}))^2 & \text{for } |y - f(\boldsymbol{x})| \leq \delta \\ \delta\left(|y - f(\boldsymbol{x})| - \frac{1}{2}\delta\right) & \text{otherwise} \end{cases} \tag{1.6}$$

  Note that the Huber loss has an hyperparameter $\delta \geq 0$ that needs to be fixed. Figure 2 (left) shows the Huber loss for different values of location parameter δ.

Example 3 (Loss functions for classification)**.** Consider a K-class classification task $\mathcal{Y} = [K]$ with vectorised data $\mathcal{X} = \mathbb{R}^d$. Ideally, we would like to minimise the number of misclassified examples in expectation. This corresponds to minimising the so called **0/1 loss**:

$$\ell(y, \boldsymbol{f}(\boldsymbol{x})) = \mathbf{1}\,(y \neq f(\boldsymbol{x})) \tag{1.7}$$

where $f : \mathcal{X} \to [K]$ is a classifier. The predictor f is discrete, and the 0/1 loss function is non-convex and nowhere differentiable. As discussed in section 1, the solution to the first observation is to define $f = d \circ \boldsymbol{g}$ with $\boldsymbol{g} : \mathcal{X} \to \mathbb{R}^K$ and d a decoder. The solution to the second consists of defining a **surrogate loss** function on g which is convex. The most popular example is the **cross-entropy loss**:

$$\ell(\boldsymbol{y}, \boldsymbol{g}(\boldsymbol{x})) = -\sum_{k=1}^{K} y_k \log \frac{e^{y_k g(\boldsymbol{x})_k}}{\sum_{k'=1}^{K} e^{y_{k'} g(\boldsymbol{x})_{k'}}} \tag{1.8}$$

where the labels are one-hot encoded.[3] A particular case which is often studied in the theoretical literature is binary classification ($K = 2$). In this case, it is common to encode the labels as Radamacher variables $y \in \{-1, +1\}$ (and hence $f(\boldsymbol{x}) = \text{sign}(g(\boldsymbol{x}))$). The most popular loss functions for binary classification are the **logistic loss**:

$$\ell(y, g(\boldsymbol{x})) = \log\left(1 + e^{-yg(\boldsymbol{x})}\right), \tag{1.9}$$

which is just a particular case of the cross-entropy loss, and the **hinge loss**:

$$\ell(y, g(\boldsymbol{x})) = \max(0, 1 - yg(\boldsymbol{x})). \tag{1.10}$$

Note that these two examples are only a function of the *margin* $t = yg(\boldsymbol{x})$. Figure 2 (right) compares the logistic and hinge losses with the 0/1 loss defined in eq. (1.7), which when written in terms of the score function $g(\boldsymbol{x})$ in binary classification is given by a Heaviside step function $\ell(y, g(\boldsymbol{x})) = \Theta(-yg(\boldsymbol{x}))$.

[3] The one-hot encoding consists of embedding the labels $y \in \{1, \ldots, K\}$ into vectors $\boldsymbol{y} \in \{0, 1\}^K$ with components $y_k = 1$ if $y = k$ and $y_k = 0$ otherwise.

1.2 Bayes risk

Given a loss function $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+$, what is the best achievable risk? Note that the population risk can be decomposed as:

$$R(f) = \mathbb{E}[\ell(Y, f(X))] = \mathbb{E}_x[\mathbb{E}[\ell(Y, f(x))|X = x]] \tag{1.11}$$

where we used the probabilist notation of Y, X to denote the random variables, as opposed to the values they can take. This defines the conditional risk:

$$r(z|x) = \mathbb{E}[\ell(Y, z)|X = x] \tag{1.12}$$

which is a deterministic function of both $x \in \mathcal{X}$ and $z \in \mathcal{Y}$.

Proposition 1 (Bayes predictor)**.** The population risk is minimised by the following *Bayes predictor* $f_\star : \mathcal{X} \to \mathcal{Y}$:

$$f_\star(x) \in \operatorname*{argmin}_{z\in\mathcal{Y}} \mathbb{E}[\ell(Y, z)|X = x] = \operatorname*{argmin}_{z\in\mathcal{Y}} r(z|x) \tag{1.13}$$

Note that although the global minimiser might not be unique, the risk associated at the Bayes predictor, known as the *Bayes risk*, is unique:

$$R_\star = \mathbb{E}_x\left[\inf_{z\in\mathcal{Y}} \mathbb{E}_Y[\ell(Y, z)|X = x]\right] \tag{1.14}$$

Given a loss function and a data distribution, the best predictor is the Bayes predictor $f_\star$, also known as *target function.* Typically, the Bayes risk is non-zero due to the noise in the labels. Of course, computing the Bayes predictor in practice is intractable, as typically we don't have access to the data distribution. Nevertheless, it provides the theoretical "gold standard" for assessing the methods we will study next.

Example 4 (Bayes predictor for the square loss)**.** Consider a regression problem $\mathcal{Y} = \mathbb{R}$ with the square loss $\ell(y, z) = (y - z)^2$. By definition, the Bayes predictor is given by:

$$f_\star(x) \in \operatorname*{argmin}_{z\in\mathbb{R}} r(z|x) \tag{1.15}$$

with $r(z|x) = \mathbb{E}[(Y - z)^2|X = x]$. Since this is a differentiable function of $z \in \mathbb{R}$, minimisers are critical points:

$$\partial_z r(z|x) = -2\mathbb{E}[Y - z|X = x] \overset{!}{=} 0 \tag{1.16}$$

This has a unique solution, given by the conditional expectation over the likelihood $p(y|x)$:

$$f_\star(x) = \mathbb{E}[Y|X = x] \tag{1.17}$$

Example 5 (Bayes predictor for the 0/1 loss)**.** Consider a binary classification problem $\mathcal{Y} = \{-1, +1\}$ with the 0/1 loss $\ell(y, z) = \mathbf{1}(y \neq z)$. By definition, the Bayes predictor is given by:

$$f_\star(x) \in \operatorname*{argmin}_{z\in\{-1,+1\}} r(z|x) \tag{1.18}$$

with $r(z|x) = \mathbb{E}[\mathbf{1}(Y \neq z)|X = x] = \mathbb{P}(Y \neq z|X = x)$. Denoting $g(x) = \mathbb{P}(Y = 1|X = x)$, we have:

$$f_\star(x) = \operatorname{sign}(2g(x) - 1) = \begin{cases} +1 & \text{if } g(x) \geq \frac{1}{2} \\ -1 & \text{if } g(x) < \frac{1}{2} \end{cases} \tag{1.19}$$

Note that what happens exactly at $g(x) = 1/2$ is irrelevant as the Bayes risk is the same:

$$R_\star = \mathbb{E}_X[\min\{g(X), 1 - g(X)\}] \tag{1.20}$$

2 Empirical risk minimisation

Now that we made sense of what learning is, and we have introduced a way of assessing how good a predictor is, it remains to establish a systematic procedure for how to find good predictors. In other words, an *algorithm* for learning.[4] While different supervised learning algorithms exist - such as decision trees and local averaging methods - our focus in the following will be on *empirical risk minimisation*, which is arguably one of the most popular supervised learning frameworks.

Consider a supervised learning problem with training data $\mathcal{D} = \{(x_i, y_i) \in \mathcal{X} \times \mathcal{Y} : i \in [n]\}$ independently from a distribution p. Let $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+$ denote a loss function. As we discussed above, ideally we would like to find a predictor which has small population risk. A natural idea for finding a predictor would be to minimise the population risk. However, in supervised learning we don't have access to the data distribution, only to a finite number of samples from it - the training data $\mathcal{D}$. Therefore, we cannot minimise the population risk directly. The main idea of empirical risk minimisation is minimise instead the empirical risk:

$$\min_{f \in \mathcal{H}} \hat{R}(f; \mathcal{D}) \coloneqq \frac{1}{n} \sum_{i=1}^{n} \ell(y_i, f(x_i)). \tag{2.1}$$

Note that minimising over the set of all functions $f : \mathcal{X} \to \mathcal{Y}$ is an intractable computational problem, and therefore one should restrict the minimisation problem in eq. (2.1) to a subset $f \in \mathcal{H}$, also known as the *hypothesis class*. Most commonly, we further restrict the minimisation to parametric function classes, i.e. hypotheses $\mathcal{H} = \{f(\cdot; \theta) : \mathcal{X} \to \mathcal{Y} : \theta \in \Theta\}$. In this case, the empirical risk can be thought as a function of the parameters $\theta \in \mathcal{D}$ instead: $\hat{R}(f; \mathcal{D}) = \hat{R}(\theta; \mathcal{D})$[5].

⚠ The choice of a hypothesis class $\mathcal{H}$ (parametric or not) introduces a bias in learning, often refereed as the *inductive bias*. Indeed, the choice of $\mathcal{H}$ reflects a belief of the statistician on how the data likelihood $p(y|x)$ looks like. For example, a choice of linear function reflects a belief that the response is linearly correlated to the covariates.

Below, we give a few examples of popular parametric hypotheses.

Example 6 (Hypotheses classes)**.** To illustrate, consider a regression task with $\mathcal{Y} = \mathbb{R}$.

- **Linear functions:** Let $\boldsymbol{\varphi} : \mathcal{X} \to \mathbb{R}^p$ denote a vector-valued function, also known as a *feature map*. The hypothesis of linear functions is defined by:

$$\mathcal{H} = \{f(x; \boldsymbol{\theta}) = \langle \boldsymbol{\theta}, \boldsymbol{\varphi}(x) \rangle : \boldsymbol{\theta} \in \mathbb{R}^p\} \tag{2.2}$$

- **Generalised linear models:** Let $\boldsymbol{\varphi} : \mathcal{X} \to \mathbb{R}^p$ denote a feature map and $g : \mathbb{R} \to \mathbb{R}$ a real-valued function. The hypothesis of generalised linear models is defined by

$$\mathcal{H} = \{f(x; \boldsymbol{\theta}) = g\left(\langle \boldsymbol{\theta}, \boldsymbol{\varphi}(x) \rangle\right) : \boldsymbol{\theta} \in \mathbb{R}^p\} \tag{2.3}$$

- **Two-layer neural networks:** Let $\sigma : \mathbb{R} \to \mathbb{R}$ be a real-valued function and consider vectorised covariates $\mathcal{X} = \mathbb{R}^d$. The hypothesis of fully-connected two-layer neural networks is defined by:

$$\mathcal{H} = \left\{ f(x; \boldsymbol{a}, \boldsymbol{W}) = \sum_{k=1}^{p} a_k \sigma\left(\langle \boldsymbol{w}_k, \boldsymbol{x} \rangle\right) : \boldsymbol{a} \in \mathbb{R}^p, \boldsymbol{W} \in \mathbb{R}^{p \times d} \right\} \tag{2.4}$$

  Note that that on a high-level, this can be seen as a linear model with a parametric feature map $\boldsymbol{\varphi}(\boldsymbol{x}; \boldsymbol{W}) = \sigma(\boldsymbol{W}\boldsymbol{x})$ which itself adapts to the data via empirical risk minimisation.

[4] Technically, and algorithm $\mathcal{A}$ can be defined as a map that takes the data $\mathcal{D}$ and returns a hypothesis $\hat{f}(\mathcal{D}) \in \mathcal{H}$ in the chosen hypothesis class.

[5] Examples of non-parametric function classes include kernel methods, which we will see later.

Therefore, empirical risk minimisation maps the problem of learning to an optimisation problem over a (random) objective function $F(\theta) := \hat{R}(\theta; \mathcal{D})$. Except for a few examples, such as linear functions, this objective function is not convex, and therefore it might have several minimisers $\hat{\theta}(\mathcal{D}) \in \text{argmin } \hat{R}(\theta; \mathcal{D})$.

⚠ The minimiser $\hat{\theta}(\mathcal{D}) \in \text{argmin } \hat{R}(\theta; \mathcal{D})$ is itself random, since it is a function of the training data. Therefore, although the test error $R(\hat{\theta})$ is a deterministic function conditioned on the minimiser $\hat{\theta}(\mathcal{D})$, it is a random function of the training data $\mathcal{D}$.

A good choice of hypothesis? — At this point, one might ask why not considering a very expressive hypothesis class $\mathcal{H}$. The reason is two-fold. First, there is a computational reason: the more parameters there are, the more parameters we have to optimise - making the optimisation computationally more demanding. Second, a statistical reason: the training data is typically noisy, meaning that the richer our hypothesis, the more likely we will fit the noise in the data (also known as *overfitting*), which might hurt the test error. Traditional wisdom suggests that a good choice for $\mathcal{H}$ is a class of functions which is neither too simple nor too complicated. It is important to keep in mind, however, that our current practice of deep learning, where neural networks with billions of parameters are both successfully optimised and achieve good test performances defy this traditional wisdom.

3 Optimising the risk: descent algorithms

Empirical risk minimisation maps the learning problem to an optimisation problem:

$$\min_{\theta \in \Theta} \hat{R}(\theta; \mathcal{D}) := \frac{1}{n} \sum_{i=1}^{n} \ell(y_i, f(x_i; \theta)). \tag{3.1}$$

Since the empirical risk above is generically a non-convex function, the choice of optimisation algorithm is important. In particular, two different algorithms can lead to two different minimisers which might have different generalisation properties. We now discuss some of the most popular optimisation algorithms used for machine learning, known as *descent-based methods*.

3.1 Gradient descent

Perhaps the most natural algorithm for optimisation is *gradient descent* (GD):

$$\begin{aligned} \theta_{k+1} &= \theta_k - \eta_k \nabla_\theta \hat{R}(\theta_k; \mathcal{D}) \\ &= \theta_k - \eta_k \frac{1}{n} \sum_{i=1}^{n} \nabla_\theta \ell(y_i, f(x_i; \theta_k)) \end{aligned} \tag{GD}$$

which consists of simply updating the weights in the steepest descent direction, with each step scaled by the *learning rate* $\eta_k > 0$. Note that GD naturally stops at a point in which $\nabla_\theta \hat{R} = 0$, which can be both a local or global minima. Defining a continuous function $\theta(\eta_k k) = \theta_k$ by piecewise affine interpolation, when the step size is small $\eta_k \to 0^+$, GD is well approximated by a continous *gradient flow*:

$$\dot{\theta}(t) = -\nabla_\theta \hat{R}(\theta(t); \mathcal{D}). \tag{3.2}$$

where $\dot{\theta} := {}^{\mathrm{d}\theta}/_{\mathrm{d}t}$. Or, seeing things in the opposite way, GD can be seen as the Euler discretisation of gradient flow with $t = k\eta_k$.

3.2 Stochastic gradient descent

A drawback of GD is that at every step k, one needs to compute the full gradient over the empirical risk. This means running over the full training set at every time - which can be slow if n is large. A simple way to avoid this computational bottleneck is to estimate the gradient at each step k only on a subset $b_k \subset [n]$ (known as a *mini-batch*) of the training data, which gives *stochastic gradient descent* (SGD):

$$\begin{aligned} \theta_{k+1} &= \theta_k - \eta_k \nabla_\theta \hat{R}(\theta; b_k). \\ &= \theta_k - \eta_k \frac{1}{|b_k|} \sum_{i \in b_k} \nabla_\theta \ell(y_i, f(x_i; \theta_k)). \end{aligned} \tag{3.3}$$

Together with its variants, SGD is one of the most used algorithm in modern machine learning.

⚠ Note that the choice of mini-batch $b_k \subset [n]$ plays a crucial role in the algorithm. Popular choices are: (a) sampling the mini-batches uniformly from $[n]$ with replacement or (b) partitioning $[n]$ over K disjoint subsets and going through the data in order. In this case, each full-pass over the data is known as an *epoch*.

Besides being computationally more efficient than GD, one advantage of SGD is that when the mini-batches b_k are chosen independently and without replacement[6], a single epoch of SGD it can be seen as an approximation for gradient flow on the population risk:

$$\dot{\theta}(t) = -\nabla_\theta R(\theta(t)). \tag{3.4}$$

Indeed, at each $k > 0$ the gradient $\nabla_\theta \hat{R}(\theta_k; b_k)$ is an unbiased estimate of the population gradient $\nabla_\theta R(\theta_k)$ (Robbins and Monro, 1951). This limit is known as *one-pass SGD*[7], and is mostly often studied in the particular case of $|b_k| = 1$. Note that this is not the case for GD or SGD with replacement, since seing the same data point (x_i, y_i) implies that the gradients will be biased.

Although the one-pass setting might seem unrealistic on a first sight, it is worth noting that it is a good approximation to certain scenarios, such as Large Language Models (LLM) like ChatGPT-3 which are trained of billions of tokens, see e.g. Table 2.2 in Brown et al. (2020).

Remark 1. In one-pass SGD with $|b_k| = 1$, each step corresponds to seeing one sample (x_i, y_i), and therefore the amount of data required to achieve a given error is equal to the number of SGD steps - or in other words: *convergence rates are equivalent to the sample complexity.*

With the observation above in mind, an useful way of thinking about one-pass SGD is as a noisy version of gradient descent on the population risk:

$$\theta_{k+1} = \theta_k - \eta_k \nabla_\theta R(\theta_k) + \eta_k \varepsilon_k \tag{3.5}$$

where we defined the effective noise:

$$\begin{aligned} \varepsilon_k &:= \frac{1}{|b_k|} \sum_{i \in b_k} \nabla_\theta \ell\left(y_i, f(x_i; \theta_k)\right) - \nabla_\theta R(\theta_k) \\ &= \frac{1}{|b_k|} \sum_{i \in b_k} \nabla_\theta \ell\left(y_i, f(x_i; \theta_k)\right) - \mathbb{E}\left[\nabla_\theta \ell(y, f(x; \theta_k))\right] \end{aligned} \tag{3.6}$$

which has zero mean since the estimation is unbiased[8]. This observation can be surprising at first sight: on average one-pass SGD optimises the true population risk even though this is an unknown

[6] When enough data is available $n \gg |b_k|$, this can be achieved by partitioning the training data in disjoint batches and running a single epoch of SGD

[7] Sometimes also refereed to online SGD, specially in the Statistical Physics literature.

[8] Note that since a fresh batch b_k is drawn at every step k, θ_k is independent of (x_i, y_i) for $i \in b_k$.

function! Therefore, understanding one-pass SGD essentially amounts to understanding two things: (a) gradient descent on the population risk, and (b) the properties of the effective noise ε_k. It is important to stress, however, that ε_k is a not a simple Gaussian noise, and therefore as a stochastic process *SGD can be very different from Brownian motion.*

4 Risk decompositions

We now have introduced the three key components involved in supervised learning:

- **The data**: The training data $\mathcal{D} = \{(x_i, y_i) \in \mathcal{X} \times \mathcal{Y} : i \in [n]\}$, assumed to be drawn i.i.d. from a distribution $p(x, y)$.
- **The architecture**: The (typically parametric) class of functions $\mathcal{H} = \{f(\cdot;\theta) : \mathcal{X} \to \mathcal{Y} : \theta \in \Theta\}$ we choose to fit the data, also known as hypothesis class.
- **The algorithm**: The practical procedure we use to choose a particular function/hypothesis $\hat{f} \in \mathcal{H}$ using the training data $\mathcal{D}$, often by minimising the empirical risk (ERM) using a descent-based algorithm such as SGD.

Successful learning crucially depends on these three factors, and therefore it is envisageable to understand how each one of them impact the generalisation error. This motivates us to decompose the excess as a function of these different components. For concreteness, consider a supervised learning problem with training data $\mathcal{D}$, and consider empirical risk minimisation with a given algorithm (e.g. SGD). Let $\hat{\theta}(\mathcal{D})$ denote the output of the training algorithm. As argued in Section 1, understanding generalisation amounts to understanding the excess risk $R(\hat{\theta}) - R_\star$.[9]

A first coarse decomposition consists of separating the excess risk in an *estimation* and an *approximation* component:

$$R(\hat{\theta}) - R_\star = \underbrace{\left\{R(\hat{\theta}) - \inf_{\theta\in\Theta} R(\theta)\right\}}_{\text{estimation}} + \underbrace{\left\{\inf_{\theta\in\Theta} R(\theta) - R_\star\right\}}_{\text{approximation}}. \tag{4.1}$$

The estimation error quantifies how far the empirical risk minimiser is from the best possible predictor in the class of functions chosen by the statistician.[10] It depends on the training data, architecture and algorithm, and typically decreases with the quantity of data available. Instead, the approximation error quantifies how far the best predictor in the class is from the Bayes predictor, and hence is a deterministic quantity that only depends on the choice of architecture and on the underlying data distribution. It is independent of the amount of data available, and quantifies a fundamental limitation of approximating the Bayes predictor with a given choice of hypothesis.

The estimation error mixes both statistical and algorithmic aspects. Letting $\bar{\theta} \in \text{argmin}_{\theta\in\Theta} R(\theta)$, it can be useful to define a finer decomposition:

$$R(\hat{\theta}) - R(\bar{\theta}) = \underbrace{\left\{R(\hat{\theta}) - \hat{R}(\hat{\theta};\mathcal{D})\right\} - \left\{R(\bar{\theta}) - \hat{R}(\bar{\theta};\mathcal{D})\right\}}_{\text{statistical}} + \underbrace{\left\{\hat{R}(\hat{\theta};\mathcal{D}) - \hat{R}(\bar{\theta};\mathcal{D})\right\}}_{\text{algorithmic}} \tag{4.2}$$

The first two terms are statistical terms: they quantify how far minimising the empirical risk is from minimising the population risk. In other words, how much generalisation we loose by empirically estimating the risk, both for the $\hat{\theta}$ and $\bar{\theta}$. Sometimes these terms are also called the the *generalisation gap*. The third terms quantifies how successful the optimisation procedure, by comparing the empirical risk of the algorithm to the empirical risk of the best predictor in the class.

[9] Note this is a random quantity, so one often considers its expectation value over the draw of $\mathcal{D}$ or aims at a high-probability statement.

[10] In general, this can be outside of the class, hence the infimum.

5 Central challenges of supervised learning

In the previous sections, we introduced the main ingredients that define a supervised learning problem: the data distribution p, the hypothesis class $\mathcal{H}$ and the most popular training algorithms: GD and SGD. It is important stressing these three ingredients - data, hypothesis and algorithm - are indissociable. For instance, the hypothesis translates a belief of the statistician about the nature of the data likelihood $p(y|x)$, and has a major impact over the behaviour of the training algorithm since it defines the optimisation landscape. The more complex the hypothesis, potentially the more complex is the landscape. On the other hand, imposing convexity at the optimisation landscape will necessarily lead to a simple linear hypothesis which can only express linear relationships between covariates and response.

Below, we list some of the main theoretical questions we would like to answer for a supervised learning task.

- **Sample complexity:** how many samples n from the data distribution p we need to achieve low excess risk with a given hypothesis class $\mathcal{H}$? What type of functions are "hard" - i.e. require a lot of data - to approximate with a given architecture?

- **Architecture design:** how to choose the hypothesis class $\mathcal{H}$ when we only have access to a finite number of samples? How rich it needs to be? For example: how wide and how deep a network needs to be to approximate a given target function?

- **Algorithmic bias:** how does the choice of algorithm will impact the generalisation? Are there architectures which are "easier" to optimise than others? How the data distribution impacts the optimisation landscape?

These questions are far from new. Indeed, a whole mathematical field, known as *Computational Learning Theory*, was developed in the 80's around formalising and addressing these questions (Valiant, 1984; Vapnik, 2013). On a high-level, the central idea in subject is introduce a suitable complexity measure over the class $\mathcal{H}$ allowing to bound the generalisation gap uniformly over any hypothesis in $\mathcal{H}$ with high-probability over the training data. Therefore, these classical uniform bounds contain at their core the intuition of a trade-off between complexity (both statistical and computational) and generalisation.

However, as we already touched Section 2, the current trend in deep learning where every new generation of state-of-the-art neural network is deeper and wider clearly defies this intuition. This discrepancy begs new ideas, and is at the core of current research in machine learning theory. The goal of these lectures is to touch some of the recent mathematical developments in this direction.

References

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020.

Herbert Robbins and Sutton Monro. A Stochastic Approximation Method. *The Annals of Mathematical Statistics*, 22(3):400 – 407, 1951. doi: 10.1214/aoms/1177729586. URL https://doi.org/10.1214/aoms/1177729586.

L. G. Valiant. A theory of the learnable. *Commun. ACM*, 27(11):1134–1142, nov 1984. ISSN 0001-0782. doi: 10.1145/1968.1972. URL https://doi.org/10.1145/1968.1972.

Vladimir Vapnik. *The nature of statistical learning theory*. Springer science & business media, 2013.