Notes and Comments on S. Mallat's Lectures at Collège de France (2019)

Deep Neural Networks: how and why.

J.E Campagne[*]

Apr. 2019; rév. 4 octobre 2023

[*]If you have any comments or suggestions, please send them to `jeaneric DOT campagne AT gmail DOT com`

Table des matières

1. Foreword

***Disclaimer**: What follows are my informal notes in French, translated into rough English, taken on the fly and reformatted with few personal comments ("NDJE" or dedicated sections). It is clear that errors may have crept in, and I apologize in advance for them. You can use the email address provided on the cover page to send me any corrections. I wish you a pleasant read.*

Please note that the Collège de France website has been redesigned. You can find all the course videos, seminars, as well as course notes not only for this year but also for previous years[1].

I would like to thank the entire Collège de France team for producing and editing the videos, without which the preparation of these notes would have been less convenient.

Also, note that S. Mallat[2] provides open access to chapters of his book *"A Wavelet Tour of Signal Processing"*, 3rd edition, as well as other materials on his ENS website.

This year, 2019, is the second in the cycle of S. Mallat's Data Science Chair.

2. Introduction to the Conference Series

During this year 2019, and in the subsequent years, we will focus on **Why** it works, rather than solely concentrating on **How** it works. There are many tutorials that provide access to the software for implementing Neural Networks (acronyms used in this course: NN or MLP for Multi-Layer Perceptron) to solve practical cases. Therefore, providing another one here would not add value. On the other hand, the *Why* is in the realm of fundamental research that we want to promote. We will see that many questions will remain unanswered.

1. https://www.college-de-france.fr/chaire/stephane-mallat-sciences-des-donnees-chaire-statutaire/events
2. https://www.di.ens.fr/~mallat/CoursCollege.html

2.1 Mathematical Domains

Of course, there is an algorithmic aspect that we will study (details, origins, etc.), but beyond that, understanding a Deep Neural Network (DNN) requires us to employ tools that cover a very broad range of mathematics:

— **Statistics**: We will be collecting data, so we start with empirical measurements from which we calculate estimators.

— **Probability**: To ensure that statistics yield precise results, we need to make assumptions about probability distributions.

— **Algorithmics**, including **parameter optimization**. In the early 2000s, we focused on convex optimization because there were convergence theorems. However, DNNs are full of local minima, so we are in **non-convex** situations where methods shouldn't theoretically work. Yet, *It works!* and quite well. New questions arise: why does it work? and in what situations does it work?

— Other branches will come into play because we will discover that DNNs will **perform well** only if they are **well-structured**. Since 2010, we have used **Convolutional Neural Networks** (**CNNs**), and when we want to understand (and analyze) why they work so well, we enter the field of **harmonic analysis**, which is related to the *Fourier Transform* and also to *multiscale analyses* such as *Wavelets*, introduced in 2018.

— These networks (DNNs) are dynamic, for example, during learning, and so they can be seen as **dynamical systems**. This is a new branch of mathematics that addresses the dynamics and convergence of complex systems.

— Lastly, another mathematical domain that will be useful is **geometry**. Indeed, these DNNs perform **high-dimensional learning**. It is necessary to understand the structures in these spaces, which means grasping the **symmetries** (representations of **groups**) of these systems.

In the end, all these fields will interact, and we will be at the frontier of current mathematical knowledge.

Another aspect, of course, underlying DNNs, is **"artificial intelligence"** (AI). Although the course does not cover it, it is clear that DNNs raise questions about the relationships/modeling of the interaction between our brain and its environment through the senses (vision, hearing) but also communication (language). This is closely related to the concept of intelligence. There is a long history of studying intelligence, and the response of DNNs is at least surprising, if not counterintuitive.

Finally, another underlying aspect of the course is **"complexity architecture"**. This is the title of an article from 1962 by Herbert A. Simon (1916-2001), "Nobel" laureate in economics in 1978 and, more importantly, Turing laureate in 1975 for his work in AI, making him one of the pioneers of AI in the USA. Why is this notion important?

The first course (in 2018) focused on the **curse of dimensionality**, in which we realized that a function with a very large number of variables (think of the millions of pixels in an image) that wants to answer a question must deal with an **extremely large combinatorial space**. And *a priori*, the number of samples required to successfully perform learning is an exponential function of the dimension, and thus seemingly hopeless. It is important to keep this in mind to understand the complexity of the problem. However, **DNNs learn with a *reasonable* number of samples**, for example, to recognize sounds, images, or language. Why? The reason is that the underlying function is not arbitrary; it has many regularities. But **what kind of regularity is it?**

In fact, if the world (problem) is very complex, there is strong structuring, and DNNs are capable of apprehending this structuring, as long as they can approximate the original function. So, by studying the architectures of DNNs, we learn about the nature of the regularities of the functions they approximate. And we ask the question: while the problem is very complex, why are the functions *also* simple (i.e., regular)? We will draw a connection with physics and the concepts of DNN architectures.

2.2 Data Analysis

We are addressing concepts that are **generic**, meaning they apply equally well to the analysis of audio signals ($d \approx 10^6$ time samples), images ($d \approx 10^6$ pixels), texts ($d \approx 10^6$ words), agents in a social network ($d \approx 10^9$ individuals), or even moles of something in chemistry ($d \approx 10^{24}$ entities). Therefore, we have a vast amount of data to deal with, and the question arises of how to find methods without specializing too much in any particular field. Two significant questions come to mind:

— **Data modeling**: This means capturing their nature and variability, understanding their structures. We can associate unsupervised learning methods (see the 2018 first course), and we will revisit them. In the potentially vast space in which data evolve, the question is: what is the restricted subspace that is actually explored? We will implement probabilistic models. The applications of modeling are diverse: for example, if we want to do **compression**, we need to understand the structure, develop models to determine the smallest number of bits possible that allows signal restoration; if we want to **restore** images, such as in medical imaging, we also need to define models; the same applies if we want to do **image synthesis**.

— **Prediction**: More related to AI, it involves using data to know the answer to a question (generally speaking). For example, what is the type of animal in a given image? In this type of problem, we also find medical diagnosis, text analysis (recognizing the author, fields/categories of the text, etc.), translation, and more. All of this falls under the domain of **statistical learning**, and it's through the work in this field that there has been something of a *revolution* in the last ten years.

2.3 Supervised Learning

2.3.1 The 2 Types of Problems: Classification/Regression

We are working in high dimension d, as previously mentioned, and we denote the vector of a sample as $x = (x(1), \dots, x(d)) \in \mathbb{R}^d$.

The first type of problem is **Classification**: Given x, what is $f(x)$, the class to which it belongs? Let's say, for example, I have images of different trees, animals, flowers, etc. The domain of classes is vast, and within each class, there is **significant variability**. In supervised learning, we assume that we have a certain number of training images/samples, meaning we have n **labeled** samples $\{x_i, y_i = f(x_i)\}_{i \le n}$. The immense variability within a class is one way to visualize the **curse of dimensionality**. To cope with this, we need to find what is common to all elements of a class (reveal their structure), which means finding the **invariants of a class**, and among these invariants, those that **differentiate the different classes**. Implicit in invariance is the notion of symmetry, and thus, the theory of groups (cf. geometry) plays a role.

The second type of problem is **Regression**, which means we want to know the value of $f(x)$ as a function, for example, taking values in $\mathbb{R}$ (rather than as a discrete class index). An example of such a problem is *Physics*. The question (a trivial one) is whether we can learn Physics without knowing the underlying "laws"[3] but "simply from the data". In quantum chemistry, x describes the position of each atom, and we would like to calculate the quantum energy of the system $f(x)$. In astrophysics, given the position, velocity, and mass of objects (planets, stars, galaxies, etc.), can we know the energy of the system? We know that if we knew $f(x)$, then by differentiating it (taking the gradient), we would have access to the "forces/interactions" of the problem and thus learn about Physics... But if I add a molecule, a galaxy, would we be able to calculate its energy without knowing the Schrödinger equation or the law of gravity? Yes, but! It requires learning with **prior information**, and it is a subject of study to understand how this *prior information* is captured by DNNs.

The case of Physics is interesting because we know many principles that constrain $f(x)$, which is not the case in classifying cats/dogs, for example, where we know nothing *a priori* about what connects an image to a dog or a cat or another object. In Physics, we can ask much more pertinent questions about *Why it works*, especially since the notion of **symmetries** is fundamental. Note that Physics in the broad sense (including Chemistry) was traditionally the discipline of high dimensionality, and *statistical learning* has also become a discipline of high dimensionality. Therefore, it is not surprising that the former can guide the path to understanding the latter.

2.3.2 Learning Algorithm

Let's take the case of an image x: we ask the question "what animal is in this image?" The answer y in this case is "dog". An algorithm in statistical learning is **parameterized** (Figure 1), potentially with a large number of parameters, such as the weights of the DNN. The learning phase allows us to **optimize the algorithm** so that the estimation **on the examples** $\tilde{y}_i$ approximates the correct answer y_i. The challenge, obviously, is **generalization**. That is, if we give an x unknown to the training set, the algorithm's response $\tilde{y}$ should be close to the true value y. Ideally, it should even be equal to it. The underlying question is: what functions will be favorable for learning? What is the **regularity** that

3. NDJE: or can we discover new relevant laws?

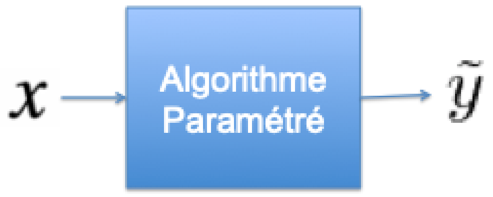


FIGURE 1 – Schematic view of an algorithm that takes input x and provides an estimate $\tilde{y}$.

these functions must satisfy so that, from a few samples, we can interpolate an answer that is a good approximation of the solution? **The parameterized algorithm above can be seen as an interpolation algorithm.**

Mathematical perspective: We approach this through regularity that reveals the structuring of the problem. However, as we saw in the first course (2018), notions of regularity through **continuity/differentiability** are not sufficient and **do not work** at all **in high dimension**.

Computer Science perspective: All the algorithms used in DNNs are very elegant and work well, but we don't understand them well. Therefore, it is a fundamental research topic.

2.4 Unsupervised Learning

We have data, and we want to understand the underlying structures: vortices in a fluid, filaments in interstellar helium/hydrogen gases. These are modeling problems, and we would like to know the probability density $p(x)$ of each realization x and, for example, differentiate between a turbulence field and another type of random field.

This type of problem (e.g., turbulence) is not new in Physics; see Kolmogorov's papers from the 1940s, and yet their resolution is still relevant. Note that Kolmogorov introduced a scale invariance model related to the Navier-Stokes equation. However, this model, even though it provides an approximation, is ultimately not suitable for the fine description of turbulent phenomena.

The problem of synthesis/modeling becomes more obscure when we are dealing not with "textures" like turbulent fields but with faces or objects, or even complete scenes mixing everyday objects. Already the problem is much less well-posed than in the case of turbulence fields because there are no underlying equations (Navier-Stokes or others). Nevertheless, it is evident empirically that DNNs are capable of addressing this type of problem (in Physics or elsewhere) without us really knowing why.

In unsupervised learning, we do not have *labels*, and it must be recognized that while there is no cost associated with data acquisition (although it depends on the domain), having labels is expensive in the industrial world. Therefore, it is natural to consider **clustering** algorithms, and this is a theme that comes up from time to time. The question then is: is it a fantasy to be able to do classification without labels? In some "simple" cases, indeed, it will work. In fact, **the number of degrees of freedom must be low** (cf. the dimensionality of the problem). It does not work in high dimensions (**curse**) because the points are far apart, and a "sphere" covering almost the entire space is needed to collect a similar sample (see the 2018 first course and the following section).

2.5 Revisiting the Curse of Dimensionality

This was the subject of the 2018 course, but we revisit it here because it is essential to grasp the problem. In supervised learning, we have labeled samples $\{x_i, y_i = f(x_i)\}_{i \leq n}$, and we seek to approximate $f(x)$. We can represent x_i as a single point in d-dimensional space and $f(x_i)$ as a color index, for example. Thus, we can study the distribution of these points in space.

Now, let x be a new point: what is $f(x)$? The immediate idea is to interpolate the value of f at point x. To do this, we need to collect the nearest neighbors of x that are labeled. The problem is that generally in **high dimensions**, $||x - x_i||$ is very large! To convince ourselves, consider the following example. We are in d dimensions in the space $[0, 1]^d$, and we want to ensure that for any x, we find a neighbor at a distance less than

1/10, meaning that the x_i must satisfy:

$$\forall x \in [0,1]^d, \exists x_i \in [0,1]^d \quad / \quad ||x - x_i|| \leq 1/10 \tag{1}$$

If the x_i are uniformly distributed (i.e., we do not favor any particular x), then 10^d points are needed to cover the entire space. However, one must consider that $d \approx 10^6$ in typical high-dimensional cases, so the number of samples is astronomical. In practice, as soon as $d \geq 20$, we are in a high-dimensional regime.

So, it is quite rare that we can use nearest neighbor algorithms; in any case, it should only be reserved for low dimensions. On the other hand, in high dimensions, if we want it to work, we will either have to discover much more global forms of **regularity** or perform **dimensionality reduction**, which can also be interpreted as the discovery of underlying regularities that allow, for example, the identification of object classes (flowers, chairs, lamps, faces, etc.). The bottom line is that having *a priori* labels is very important in high dimensions, and yet it is the most costly.

2.6 Linear Classifiers

These concepts were covered in the 2018 course, and they date back to the 1990s. The idea is to simplify the problem as much as possible by linearizing it. What we want is to find a transformation (or change of variables) Φ such that we can **linearize the boundary** (see Figure 2). To illustrate, let's consider the case of a 2-class classification:

$$x = (v_1, \ldots, v_d) \xrightarrow{\Phi} \Phi(x) = (v'_1, \ldots, v'_d) \tag{2}$$

If the technique of *linearization* is common in math/physics, here x is in a high-dimensional space, and understanding the nature of this change of variables is more complex. Once the transformation is done, the separation surface is a hyperplane with a normal vector w (see Figure 3).

Learning in this case involves determining the vector w. The classifier has a simple

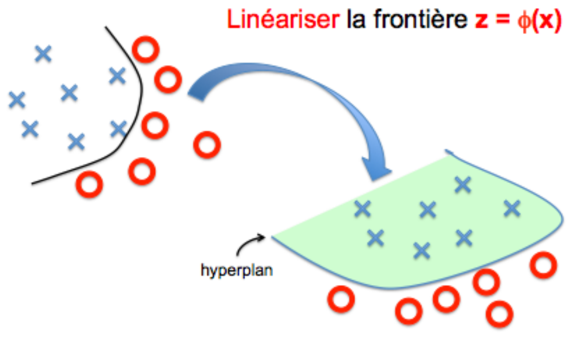


FIGURE 2 – Illustration of the use of a transformation $x \rightarrow \Phi(x)$ to place oneself in a space where the binary classification problem is separable by a hyperplane.

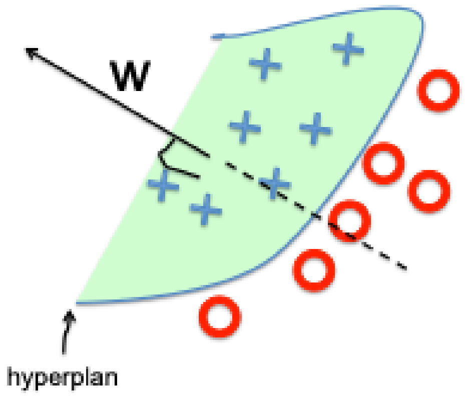


FIGURE 3 – Diagram of the normal projection to the separating hyperplane.

expression:

$$\tilde{f}(x) = \text{sign}\left\{\langle w, \Phi(x)\rangle + b\right\} = \text{sign}\left(\sum_k w_k v'_k + b\right) \tag{3}$$

A trivial example of this type of classifier is one that tells you whether there is a firetruck or a car in an image. The diversity within these 2 classes can be very high (e.g., d is the number of pixels in the image), yet counting the number of red pixels is very discriminative.

In less trivial cases, the v'_k are often called **patterns** or **features**, and although this is not the only possible interpretation, the sum $\sum_k w_k v'_k$ can be seen as a *vote* among the patterns.

So, this linearization procedure simplifies the problem greatly, but in what situation can it be applied? Or in which case can we find the transformation $\Phi(x)$?[4] Two scenarios then arise:

— either we have *prior knowledge* (e.g., firetrucks are big and red), and in this case, the transformation $\Phi(x)$ is known;

— or we have **no *prior knowledge*** (note that this is the majority of cases), and we will need to "learn" $\Phi(x)$. Neural networks provide a strategy in this case where both $\Phi(x)$ and w are **determined simultaneously** during the learning phase. However, we will see that we do not start entirely without prior knowledge, but we will work with all the variables available in dimension d[5].

2.7 Neural Networks (NN)

Computer neural networks date back to the 1950s and were modeled by analogy with the brains of small animals. The single-layer model was presented in the 2018 course, corresponding to the schema[6] shown in Figure 4. For the non-linear element, one can think of a **Rectifier** (**ReLU**), which as we will see, is widely used in deep neural networks

4. Note in passing that finding w is not the problem because there are many algorithms in linear algebra (see SVM, Logistic Regression, etc.) that deal with this topic.

5. NDJE: However, Kernel methods allow us to bypass the transformation itself since in this case, it is "enough" to apply a similarity function K; but it remains that K is a *prior choice*.

6. Note that the sign of b has changed compared to the 2018 illustrations.

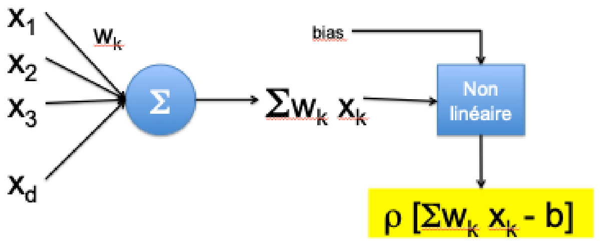


FIGURE 4 – Diagram of how a neuron operates with a part for *linear aggregation* and a part for *non-linear activation.*

(DNN):

$$\begin{cases} 0 & \text{if} \quad \sum_k w_k x_k < b \\ \sum_k w_k x_k - b & \text{otherwise} \end{cases} \tag{4}$$

The thresholding by the bias b allows for **sparse activation**[7].

It is worth noting that this single-layer model is of the same type as the linear classifier with the boundary determined by a hyperplane characterized by the w_k and bias b. Now, we can **chain layers of neurons together to form a network** (NN or MLP for Multi-Layer Perceptron) as shown in Figure 5 (2 hidden layers). We can see that the **function** $\Phi(x)$ we were looking for previously is the result of **learning the upstream part of the neural network**; the final operation is similar to that of a single neuron in Figure 4.

What does it mean to **learn a neural network**? It involves determining its parameters (the weights between different layers) by minimizing the error between $\tilde{y}_i$ and the true labeled value y_i. For this purpose, **cost functions** are defined, and the goal is to find their minimum. This is a **challenging optimization problem** because there are numerous local minima that must be navigated to find the global minimum or at least approach it[8].

It is essential to understand that there is **prior information**, as mentioned in the previous section, embedded in the **architecture of the network**. *This information is not discovered during training at the moment* although the weights can, in a way, provide

7. Note: this effect is similar to what is found in Wavelet Analysis.
8. Note: Recent studies show that solutions found by Gradient Descent methods are not arbitrary and that networks obtained in this way yield similar results.

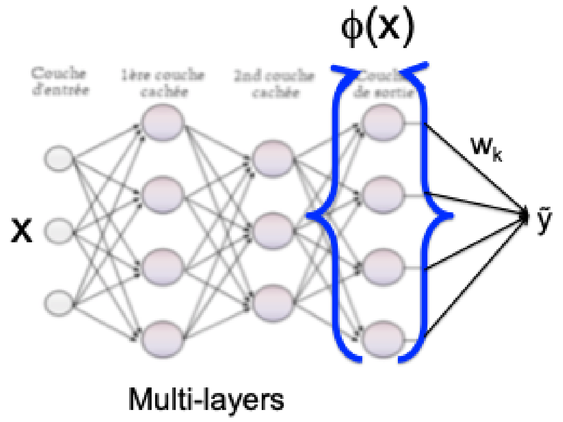


FIGURE 5 – Simplified diagram of a multi-layer neural network where the representation Φ is the result of all the sub-layers; it is used at the end of the network for tasks such as classification.

an indication of the most active neurons and those with nearly zero activity. Does this arsenal work in general? The answer is no! In other words, if you add **lots of neurons and lots of interconnected layers**, and you try to optimize them blindly, **it will not yield any useful results at all**. Therefore, only specific architectures (prior information) will produce satisfactory results. Empirically, we have discovered **relevant architectures** for certain problems, and underlying this is the idea that these architectures have effectively captured (reduced) the complexity of the problem.

2.8 Convolutional Neural Networks (CNN) or Deep Convolutional Networks (DCN)

A significant breakthrough was made in the 1990s by Y. LeCun (AT&T Bell Lab.) and J. Bengio (Univ. Montreal) in the way neural network architectures are designed[9]. They incorporated problem symmetries, such as **translation invariance** (*note: local distortions are also considered*), by using **convolutional filters** that capture this prior knowledge,

9. NDJE: See http://yann.lecun.com/exdb/lenet/index.html for a history of the first convolutional networks, which were used to automatically recognize handwritten digits on checks. Note that Y. LeCun, J. Bengio, and G. Hinton received the 2019 Turing Award.

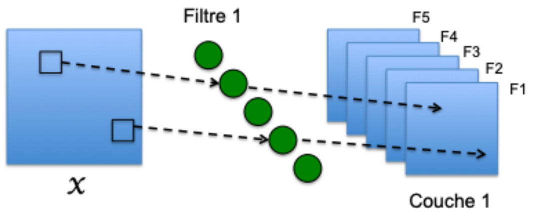


FIGURE 6 – Schematic of a first convolutional layer with 5 different filters. For each filter, e.g., filter F1, each neuron handles a small part of the original image, and all weights associated with each neuron are identical across neurons.

significantly reducing the connections in the first layer(s) of the neural network. This, in turn, simplifies weight optimization through learning.

Let's gradually dive into the details. For the sake of illustration, consider an image with $n \times n = d$ pixels. However, keep in mind that, as indicated in LeCun and Bengio's article, this procedure also applies to speech samples and time series data. First, **each neuron** in the first layer (the green bubbles in Figure 6) is **only connected to** (or responsible for) a subpart of the original image, typically a small 3×3 or 5×5 patch, for example. **Each neuron** of this type, therefore, performs a **local linear combination** of the pixel values to which it is connected (x_i with i restricted to a region), and it is also associated with a **rectifier** (or another non-linear function).

Now, imagine that in the image, we want to recognize a small object (e.g., a stick), and we don't know *a priori* where it is located in the image. Therefore, there is no reason to favor the response "a stick" in one or the other of the two patches in Figure 6. **Consequence: the weights w_k of the two neurons must be the same, reflecting translation invariance.** This leads to a **considerable reduction in complexity**: firstly, each neuron handles a small number of inputs, and secondly, the weight matrix is the same for all neurons. If we now have a **family of filters**, each specialized for a particular type of processing, the results are manifested as separate outputs, denoted as $F_1, F_2, \ldots, F_5$ in Figure 6.

Next (Figure 7), the neurons in the next layer still focus on a small patch, but this time, they consider the same area in different filtered images (see the small dashed tube). **Each neuron performs a linear combination of the results from the first layer in a small**

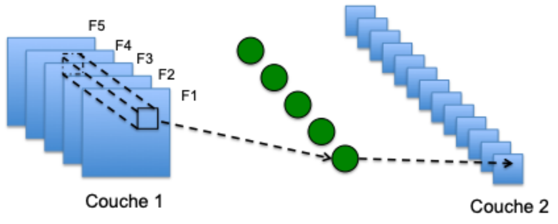


FIGURE 7 – Schematic of the second convolutional layer: the results from each filter in the first layer are combined in a new convolutional process.

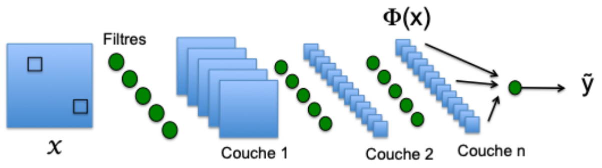


FIGURE 8 – Overview of the convolutional network with an aggregation step at the end to produce $\tilde{y}$ from the learned representation $\Phi(x)$ during the training process.

cube. By the same argument as before, we conclude that **for another neuron** that focuses on another small cube, **the weights are the same**. This process eliminates spatial location information in the original image. However, it constructs a small image in layer 2, and by varying the weights (changing the filter), a series of small images that make up layer 2 is obtained (*note: each linear combination is associated with a rectifier*).

We can continue **by stacking filtering layers to build the network architecture (CNN)**. In the end, we usually connect to a single neuron (more commonly, we have a fully-connected "classical" network) that provides the estimate $\tilde{y}$, and the last layer actually gives the transformation (change of variables) $\Phi(x)$ (Figure 8). In the end, the initial complexity of the problem has been significantly reduced because neurons are specialized, and translation invariance has been implemented by minimizing the number of weights at each layer. However, by stacking layers, we quickly end up with **CNNs with several hundred million parameters** to optimize.

The number of parameters is much larger (in general) than the original dimensionality d of the input space x. The network is then trained with typically thousands or even millions of samples, and it shouldn't work, theoretically. **However, the big surprise is that it does work**: the results on "classic" benchmarks are quite remarkable and *generic*, meaning the same architecture can tackle a wide range of problems, such as image classification, speech recognition, translation, and physics problems. We will analyze ***how it works*** in our classes, as there is a certain algorithmic art with optimization methods like stochastic gradient descent, but we will mainly focus on the ***why it works***.

Let's empirically see what happens. Neurons are capable of **significantly reducing the dimension of the problem**. For example, if we start with a sample in a 10^6-dimensional space (e.g., the number of pixels in an image), **the last layer typically has around 100 variables**. Moreover, when we analyze what happens in the layers, **the boundaries between classes flatten** (linearize), and in the end, we have a hyperplane problem for classification. But why does this flattening occur? **The architecture, of course, plays a crucial role, and this connection is what we will study through harmonic analysis and geometry (group theory).**

2.9 Course Outline for 2019

2.9.1 Phase 1: Developing Intuition and Asking Questions

We will start with the applications because, at this point, we are beyond the stage of questioning whether all of this is just *buzz* or a passing trend. The results are impressive, and there are significant industrial implications. Therefore, it's essential to develop an intuition for these complex objects by understanding how they work. Along the way, we will raise the **mathematical questions** that arise.

S. Mallat mentions that he will explore a vast universe in which each expert from a discipline can contribute to understanding the problem (*using their own tools*). Thus, we will examine various perspectives that shed light on understanding deep neural networks. We will go back to the origins of questions in Artificial Intelligence, including *Cybernetics* (Wiener 1947, Herbert Simons 1960) and the early networks of Frank Rosenblatt (1957 Perceptron). The questions raised from this early period are fundamental.

2.9.2 Phase 2: Single Hidden Layer Networks

Next, we will delve into networks with two layers (i.e., **one hidden layer**) because this is a case that we can understand reasonably well now. We are aware of the famous **universal approximation theorem** by Cybenko (1988) [10]. This theorem essentially states that **this type of network can approximate almost any function**. However, we know that this **result can be deceptive**, as if the **convergence** is guaranteed by the theorem but is **exponentially slow** in practice, it doesn't help us much.

So, we will need to study the theory of approximation because the problem is not just about making the error tend to zero; it's about making it tend to zero **rapidly enough** to be effective with the number of labeled examples we typically have.

2.9.3 Phase 3: Multi-Layer Networks

Afterward, we will move on to **multi-layer networks** (MLP: multi-layer perceptrons or DNN: deep neural networks). We will look at approximation performance, but we will quickly **hit the limits of mathematical tools**. We will then shift to the **algorithmic perspective**. We will explore optimization through **stochastic gradient** methods. The idea is old (Robbins and Monro 1951), but there has been a lot of research in this area (see the 2018 seminar on *lazy* versions). These algorithms are particularly effective for training multi-layer networks through the **backpropagation** method (Rumelhart, Hinton, Geoffrey; 1986).

2.9.4 Phase 4: Convolutional Networks (CNN)

Next, we will study **convolutional networks** (CNN or ConvNets), introduced by Y. LeCun in 1990. These networks have delivered spectacular results in human handwriting recognition and other domains. We will dive into the realms of *mathematics*, *algorithms*, and *applications*, taking our time to advance, identify underlying questions, and attempt to answer them. This course is not for those in a hurry who want to program right away;

10. NDJE: In 2018, S. Mallat mentioned it at the end of his series of lectures, postponing its proof to this year.

there are plenty of tutorials available online for using tools like Torch, Keras, TensorFlow, etc.

NDJE: This fourth part was postponed to 2020.

3. Applications

An experiment conducted by Thorpe *et al.* in 1996[11] showed that for a human, determining whether there is an animal in an image takes 150 ms, despite the diversity and complexity of the images. This means that the response is efficient but relatively slow, especially when passing through typically 10 neural layers. Furthermore, observations tend to indicate that there are no feedback loops because that would make it even slower.

3.1 Computer Vision

3.1.1 Image Classification

The first significant application was in **computer vision**, which started in the 1970s. Typically, for an image like the one in Figure 9, one would want to describe the scene, such as recognizing a football game, identifying the position of the players, and locating the position of the ball.

In the 1970s-1990s, the approach to answering these questions involved recognizing *patterns*, understanding the specific nature of an object (e.g., the shape of a ball, the features of a player) and then trying to extract this information from the image. The process often began by extracting **edges** (which could be achieved using filters), as shown in Figure 9 (right), processed using Mathematica EdgeDetect. Many papers from the 1970s-1980s described how to best detect edges. This reduced the number of relevant points significantly, making it "easier".

However, when it came to **recognizing objects** (patterns) and understanding how they are **arranged relative to each other** in the image, things became challenging, especially in complex cases. Defining metrics for comparing edges became a problem. Questions

11. Nature, Jun 6;381(6582):520-2.

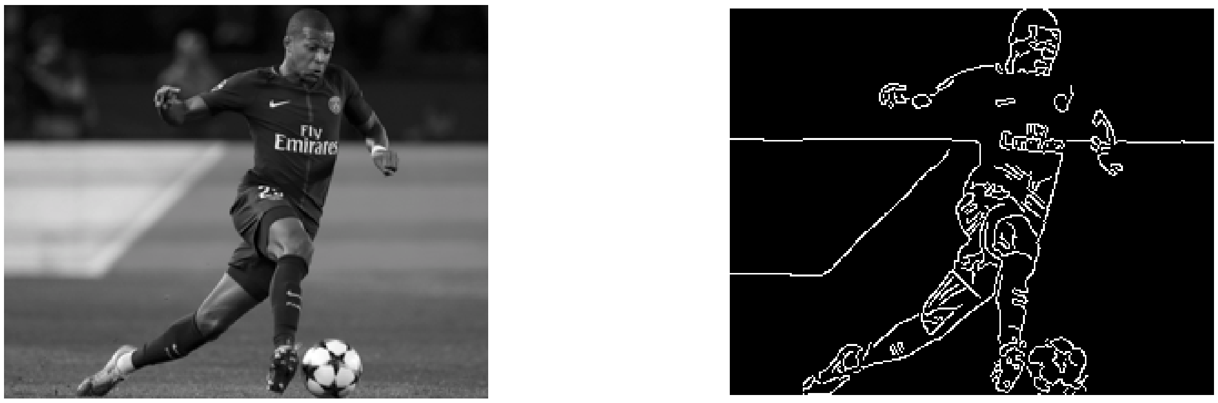


Figure 9 – Illustration of the use of "classical" image structure analysis techniques: here, edge detection.

arose, such as what defines a shape, how to describe it, and how do "objects" interact in a broad sense? These issues were examined from the perspectives of **symbolic representations** and **image grammar** (e.g., the sky is usually blue and at the top of an image, feet are at the bottom, the round head is near the top, etc.). In fact, this entire approach to image recognition was heavily **influenced by the grammar of natural language** (*cognitive science*, S. Mallat cites Noam Chomsky, the founder of generative linguistics). However, while these methods worked in simple cases (e.g., recognizing a face with a few key points), they were entirely inadequate when dealing with even slightly complex images.

There was a **failure** in this approach, and it was not easy to identify, especially given that almost the entire research community in this field had been working within the same *methodological framework* for about thirty years. It should be noted that the adoption of Neural Networks to tackle these problems marked a radical and not immediately accepted departure from the previous approach.

Neural Networks made a significant breakthrough in this field in 2012, after nearly 25 years of research. It resulted from the **ImageNet competition**, a database containing 1 million images and 2000 object classes established by researchers at Princeton. The competition aimed to determine whether **classification could be achieved** by learning from these images. At that time, the dominant algorithms for learning were **kernel-based** (Kernel methods: Kernel PCA, Kernel SVM, etc.), meaning that **features** were established *empirically* through image preprocessing and then provided to the algorithms. The field

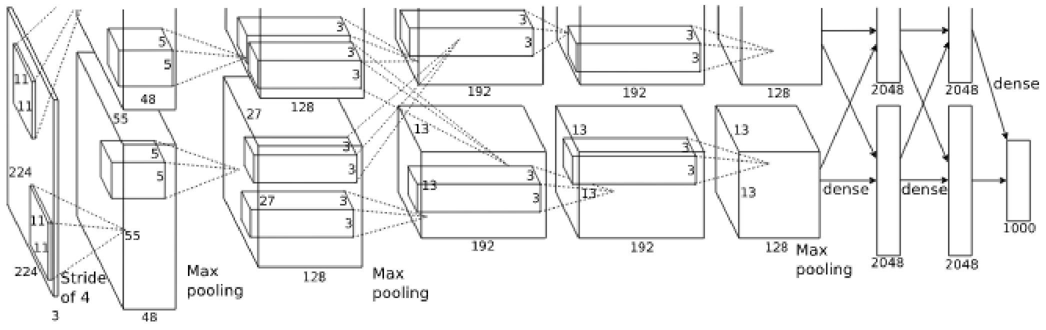


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

FIGURE 10 – Architecture of AlexNet (2012), the first neural network to beat kernel methods in the ImageNet data challenge.

saw significant progress between 2000-2010 with these techniques, which yielded good results. However, the **ImageNet problem remained very challenging**.

Then, a Canadian team led by Geoffrey Hinton (Alex Krizhevsky and Ilya Sutsever) reported their score with a network architecture presented in 2012 (submitted to NIPS 2012), the structure of which, called **AlexNet**, is shown in Figure 10.

AlexNet is a **convolutional network** with 5 layers, trained on 1.2 million images (224 x 224) containing 1000 classes. The first layer of filtering (48 different filters) consists of neurons, each handling 11x11 patches. The second layer subsamples, and the process continues with 128 filters and so on. In the end, there is a dense classifier ending in a vector with 1000 variables that provides the classification result.

It took many trials to arrive at this architecture, but the remarkable result is that it works! They achieved approximately 17% error [12], which was about a 30% improvement over the competition (see Figure 11). AlexNet became a benchmark and is relatively easy to reproduce with modern libraries. It should be noted that the best networks today have 150 layers and achieve less than 3% error, surpassing human performance (as discussed

12. NDJE: S. Mallat mentions a figure of 15.3%, which applies to a variant of the presented architecture.

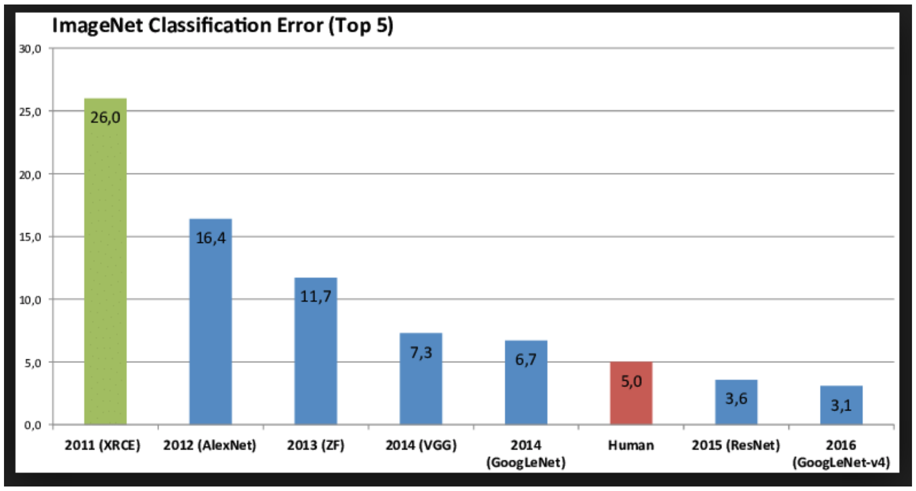


FIGURE 11 – Evolution of error rates on ImageNet for different algorithms: AlexNet in 2012 marks the beginning of the era of Neural Networks.

below). Why does the network size need to increase? It's not necessarily to capture more information but rather due to optimization issues, which we will come back to.

Why does it work? Even though the images are complex! It is believed to work so well because the network has been able to capture (learn) underlying structures, *invariants*. What are these invariants? **What is the nature of the knowledge extracted by the DNN?** These questions are important because **if we don't understand** how this type of network functions, we use it as a **black box** without being sure whether the answer is correct or not. Furthermore, in some cases, **we must be able to explain the result**, for example, when selecting job candidates, disputes may arise; in medical diagnostics, we may need to guide toward the right treatment or not...

Error Evolution: Figure 11 shows the Top 5 errors (i.e., comparing the true label to the top 5 labels with the highest probability) over the years since 2011. Before AlexNet, the best results, around 26%, were achieved with k-NN (k-nearest neighbor) or NCM (nearest class mean classifier) algorithms with very well-structured local descriptors. Then, AlexNet marked the start of neural networks with refinements in architecture and learning algorithms[13].

13. NDJE: And the accompanying hardware...

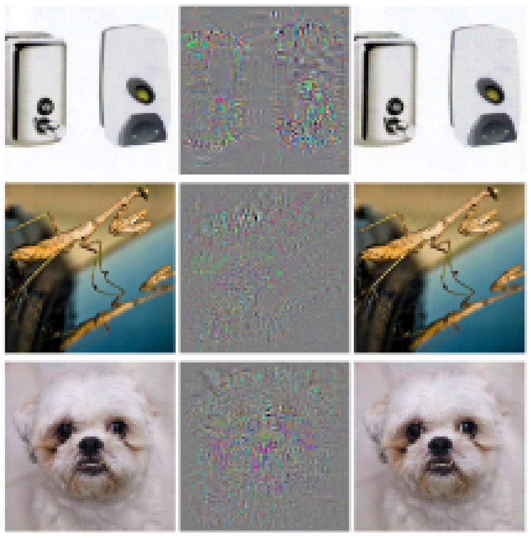


FIGURE 12 – Illustration of adversarial examples: here, AlexNet fails to recognize the "noisy" images on the right, classified as "ostrich," whereas without noise, it classifies them correctly. The noise is highly specialized, and its structure is revealed (middle images) when magnified by a factor of 10. In L2 norm, this noise represents only 1% of the signal.

It can be observed that **recent results show performance "superior" to that of a human**. However, this statement should be taken with caution, as the nature of errors is different. It's clear that a human seeing an image of an animal or an exotic flower for the first time may not recognize it, but when shown the same animal or flower in another image, humans have a much superior learning ability compared to neural networks, which require "many" examples to learn. On the other hand, in some cases, the errors made by neural networks are surprising, so **the nature of their errors is at least different from that of a human**.

Now, understanding why a **complex system makes errors** is essential, not only intellectually but also because it is connected to its stability/instability in **engineering**. This problem can be seen with neural networks through what are called **adversarial examples**. In Figure 12, the left column consists of correctly labeled ImageNet images (x), the middle column shows error images to which ϵ is added to "fool" the neural network (in this case, AlexNet) with $||\epsilon|| < 10^{-2}||x||$ (*the defects are imperceptible to the naked eye,* *and they need to be magnified by a factor of 10 to "see" anything in the middle column*). The images in **the right column**, created by $x+\epsilon$, are all identified as a **species of ostrich!** There is clearly an **instability that we absolutely want to avoid**, and therefore, we must understand it to find a solution. In fact, the nature of these instabilities is understood. In the example above, the noise was chosen specifically to cause the network to fail. A first reaction might be to think that encountering noise that reproduces these particular defects has an extremely low probability, so we can forget about it. However, **imagine that instead of classifying images, this type of neural network is in control of a commercial airliner: do we have the right to allow such catastrophic defects to occur?** The answer is clear.

However, avoiding these defects actually harms the network's performance. There is a trade-off between performance and stability. Therefore, the idea is to ensure that the **probability of encountering these "catastrophic" defects is extremely low**. Note that these "pathological cases" are counterintuitive because they **contradict the very good generalization ability of these neural networks once trained**.

3.1.2 Video Classification

In the online video, an example of which is shown in Figure 13, you can appreciate the results of a Convolutional Neural Network (CNN) classifier operating in real-time to recognize the type of sport being played. In fact, the classification is performed frame by frame. The images are complex, and the number of classes is high, yet it works remarkably well, achieving up to 80% for the Top 5 classes.

3.1.3 Other Types of Problems: Image Segmentation

Finding the contour of an unknown object in an image, for example, a Dalmatian in a black and white speckled image (Figure 14). We can see that to solve such images, one needs to have prior knowledge of what a dog is or a specific organ in the case of a medical image. This low-level problem, known as segmentation, is at the interface with structural recognition problems. This type of problem is crucial in fields such as medical image interpretation and, of course, autonomous driving. It works well with neural networks, as shown in the example in Figure 15.

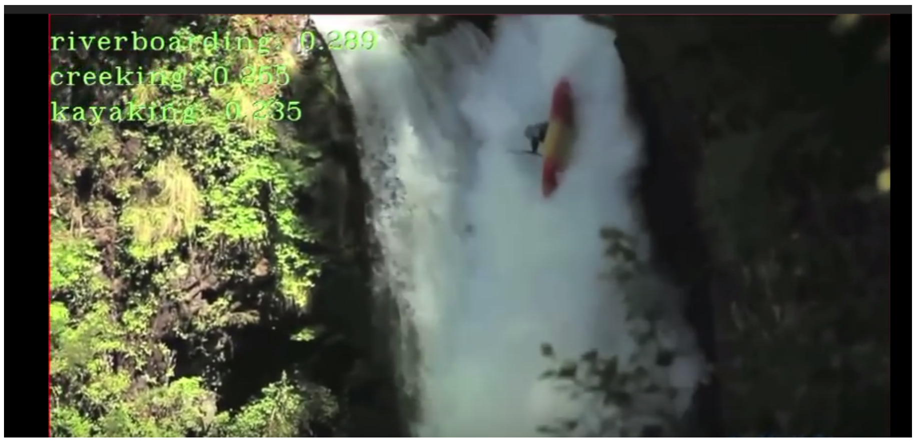


Figure 13 – Screenshot from a video (https://www.youtube.com/watch?v=qrzQ_AB1DZk) showing the results of a CNN with the top 3 sports categories and their scores in the upper left corner. Excerpt from the publication https://cs.stanford.edu/people/karpathy/deepvideo/: *classification using a new dataset of 1 million YouTube videos belonging to 487 classes.*

The network recognizes both structures and contours: low-level information (the contour) interacts within the network with high-level information (the type of structure). This problem has been a research topic since the 1960s (cf. Roberts cross operator), where the question was whether to take on the problem: for example, should one recognize "transitions" to define a contour that, through its shape, gives the structure of the object? Or should one start with a type of object that one is trying to recognize to determine its contour? In fact, CNNs allow feedback loops that mix both perspectives.

These segmentation algorithms are implemented in chips that allow real-time usage for autonomous vehicles. Videos on this subject can be found. While CNNs take a long time to train, once this step is completed, the response to a new stimulus is very fast.

However, in the case of autonomous vehicles, there are rare cases of crashes, such as "a child chasing after a ball running onto the road". Manufacturers must take them into account. Typically, these are cases that were not part of the training videos. A strategy must be found: "the problem must be structured". Not only by structuring the training database but also the architecture of the network (e.g., dividing it into specialized parts).

So, it is not enough to provide any CNN with data and hope that it converges in

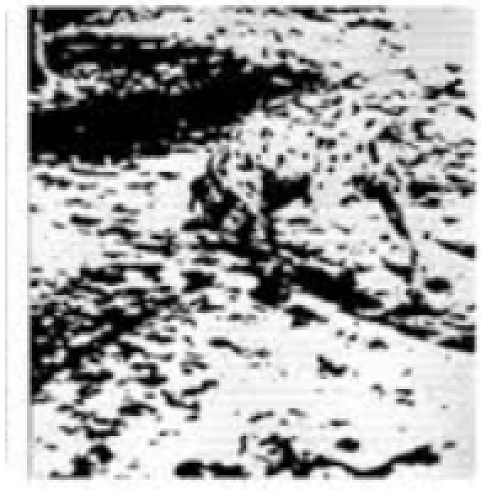


Figure 14 – The type of image that poses a problem: should one start from "contours" to guess what’s in the image, or start from "a priori shapes" and look for them in the image?

three days.

3.2 Speech Recognition

3.2.1 Traditional Approach

This is a field that dates back to the early 1960s, and the techniques used have evolved relatively little until the 2010s. Typically, the process began by taking the spectrogram of the temporal signal, as shown in Figure 16. In the top spectrogram, you can observe phonemes (from the word "encyclopedias" pronounced in English) with "fundamental + harmonic" structures, and others that are not "voiced," like "cy". So, locally in time, the recognition of a phoneme (or its elementary component) is attempted, represented by a state in a **Markov chain** (blue blobs in the figure 16). To learn these states, Gaussian Mixture Models (GMM) are used. Then, this Markov chain is optimized, taking into account the timing of phoneme occurrences. There are more probable phoneme successions in the lexicon of the speaker’s language. Therefore, **transition probabilities**

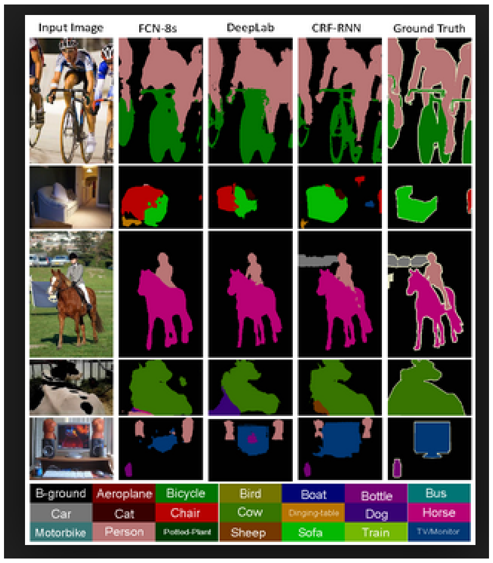


FIGURE 15 – Example of segmentation with a CNN.

(curved arrows) between a sound at $t = i$ and the next sound at $t = i + 1$ are optimized.

The technique used is a dynamic programming algorithm (**Viterbi**). In essence, this algorithm dates back to 1967 and is used to correct errors in a noisy channel. If n is the size of the noisy message and a is the number of possibilities per letter, then the possibilities tree grows as a^n. A. Viterbi found a way to simplify the tree as it is constructed. **This method was the framework within which people worked for about fifty years, and the field had become more of an industrial problem rather than academic research.**

It's worth noting that the two spectrograms in Figure 16 are, in fact, the same word ("encyclopedias") pronounced by a female speaker (top) and a male speaker (bottom). Thus, you can see the challenge of speech recognition as the diversity of voices is vast and leads to very different spectrograms.

3.2.2 The (Very Recent) Revolution of CNNs

Around 2013-14, the methodology changed. It was realized that the initial steps (Gaussian Mixture Models, Markov chain) could be replaced by Recurrent Neural Networks (RNNs), and gradually, all the steps were introduced into RNN architectures. Mo-

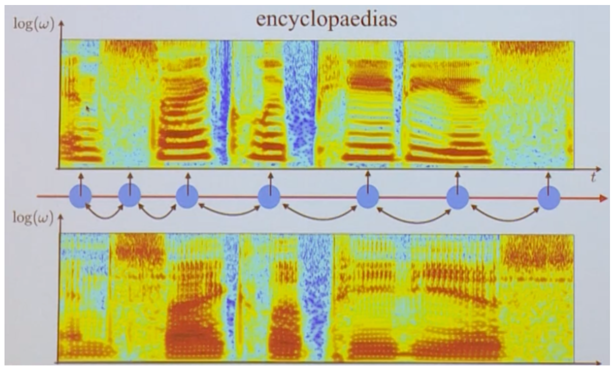


FIGURE 16 – Examples of two spectrograms from two English speakers (a woman for the top one and a man for the bottom one) recorded during the pronunciation of the word "encyclopedias". A Markov chain analysis relates phonemes to the state of the chain.

bile applications are equipped with these types of RNNs embedded in chips. The structure is similar to that for image analysis, i.e., **cascades of convolutions and subsampling**. The difference is that the filters no longer operate in the image space but in the space of **temporal series**, where the concept of **causality** is also an important ingredient.

The evolution has been rapid, and it continues. One of the classic problems, for example, is **source separation**. These developments are important, especially when considering hearing aids for the hearing impaired. If the device simply increases the sound power, it not only increases overall noise but also fails to better capture what the speaker is saying.

A "normally constituted" human, with two ears and focusing attention on the speaker, naturally performs component separation and selective amplification. In the case shown in Figure 17, the problem is initially a bit more complex because there is only one microphone, and there is a mixture of two speakers. In fact, the methods developed before 2018 worked more or less with sophisticated techniques of sparsity and separation, such as **Non-negative matrix factorization**, but they had their limitations.

The issue is that the spectrogram of the recorded sound is a mixture, as seen in Figure 17, where there are two speakers (red & blue). Ideally, each point in the **time-frequency** plane (coded here with 256 channels) is associated with either the "blue" speaker or the "red" speaker, as in a **classification** problem. However, we do not know the number

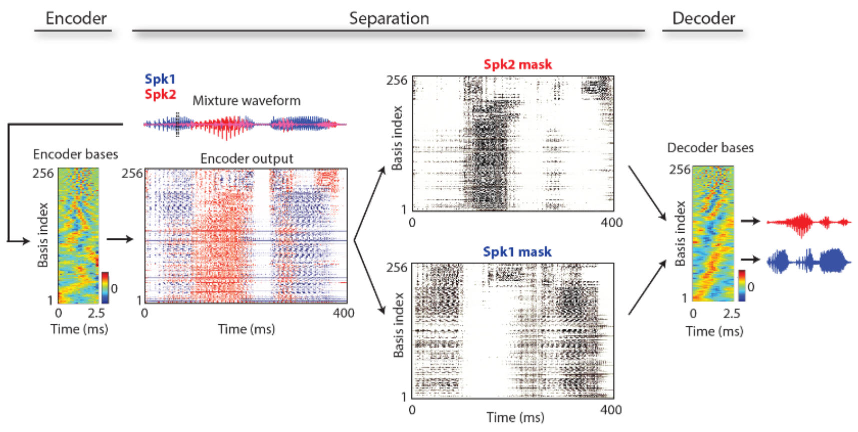


Figure 17 – Illustration of how one can proceed to separate the sound mixture of two speakers.

of speakers or the typical spectrograms of each. If, on the other hand, we can separate the speakers and have some phase information, we can reconstruct each speaker's voice, as shown in Figure 17.

However, the paper by Yi Luo and Nima Mesgarani from September 2018 (arXiv:1809.07454v2) describes the architecture of a CNN (Figure 18). The authors were able to show that **the convolutional neural network significantly outperforms methods for separating spectrogram mixtures of each speaker in the Time-Frequency domain** [14]. In this network, in fact, the "masks" of the speakers are learned at the same time as the sounds they pronounce. This follows the same pattern as deep neural networks: learning the representation simultaneously with classification. Note that S. Mallat did not think that we would reach this level of listening quality.

Well, we understand the algorithmics, but understanding the structures that have been captured is still not fully understood. We have some ideas based on previous methods that tried to take into account timbre, the frequency rigidities inherent to a particular speaker, but by using these specificities, no one has come close to what CNNs achieve.

It's worth mentioning that the authors also tried to completely do away with spectrograms, meaning they used raw recordings. They found that the initial layers of the

14. The authors had developed an LSTM network, but the one in Figure 18 is even better

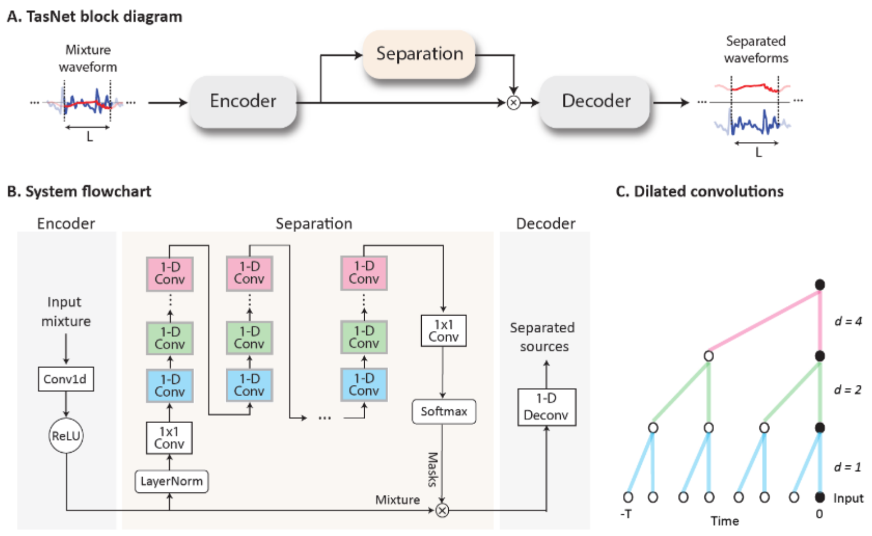


FIGURE 18 – CNN architecture from arXiv:1809.07454v2 for solving the problem of separating the voices of multiple speakers speaking simultaneously. Here is the paper's legend: (A): the block diagram of the TasNet system. An encoder maps a segment of the mixture waveform to a high-dimensional representation, and a separation module calculates a multiplicative function (i.e., a mask) for each of the target sources. A decoder reconstructs the source waveforms from the masked features. (B): A flowchart of the proposed system. A 1-D convolutional autoencoder models the waveforms, and a dilated convolutional separation module estimates the masks based on the nonnegative encoder output. (C): An example of causal dilated convolution with three kernels of size 2.

network reproduced the equivalent of a spectrogram. But beyond these initial layers, it is very difficult to understand what the CNN is doing or learning.

3.3 Natural Language Processing

3.3.1 Pre-1990 View of the Problem

This is a field that covers a wide range of areas: translation, text analysis, conversation between people, question answering, descriptive comments, and more—all related to **linguistics**. Before we take a mathematical/algorithmic perspective on the subject, let's take a completely different look that sheds light on the notions that ultimately concern how language has emerged and how it is structured. This approach will equip us with insights into algorithms armed with this knowledge.

The first grammars date back to the 16th century, and it's worth noting that the *Port-Royal Grammar*[15] published in 1660 is inspired by René Descartes' *Rules for the Direction of the Mind.*

Linguistics itself emerged in the 20th century with Ferdinand de Saussure (1857-1913), a precursor to **structuralism in linguistics**, and the development of the science of communication sign systems (semiotics) invented by Charles Sanders Peirce around the same time (1839-1914). **Structuralism** (e.g., Claude Lévi-Strauss in anthropology, Roland Barthes in literature, Jacques Lacan in psychoanalysis, Michel Foucault and Louis Althusser in philosophy, and more) distinguishes between the *signifier* (what supports the sign, e.g., sound) and the *signified* (e.g., the idea, the underlying concept) and **attempts to analyze these *signs as a system* and seeks to highlight *structures***[16]. So, we have a succession (hierarchical) of structures: sounds, phonemes, morphemes, morphology, syntax, and more. For example in French, "chanteurs" can be broken down into "chant" (the action), "eur" (the one who does it), and "s" carrying the plural meaning; "chant" belongs more to the lexical field, while "s" belongs to the grammatical field. All these concepts emerged and were guided by empiricism by segmenting and structuring discourses. **This structuralist approach dominated linguistics until the 1960s**, but **it influenced other disciplinary fields** such as image processing (segmentation into elementary structures, contours, and relationships between patterns, and more). Subsequently, post-structuralism in the 1960s and 1970s (e.g., Jacques Derrida, Michel Foucault, Gilles Deleuze, etc.) generalized these concepts and dissected social phenomena through the structure of the context from which they emerged (e.g., the structure of language).

In 1957, **Chomsky's formal grammars** emerged (Noam Chomsky (1928-), the founder of generative linguistics, "Syntactic Structures"), which means that a **more rationalistic view replaced the previous empirical view**. The concepts put forward by Chomsky have a profound connection with programming, compilers, computability theory (mathematical logic), and natural language processing. **Chomsky's hypothesis is that a part of**

15. its full title: *Grammaire générale et raisonnée contenant les fondements de l'art de parler, expliqués d'une manière claire et naturelle*

16. In doing so, it even eliminates speakers/authors, but that's another debate.

the structures of human language, its grammar and syntax, is related to the biological transmission of genetic code. From a syntactic (or grammatical) point of view, one can distinguish between *correct* and *incorrect* sentences. These grammatical structures are at the core, according to Chomsky, of a human's ability to structure all language. So, the question arises: what are the structures of grammars in general, and to what type do natural language grammars correspond? According to Chomsky, there are **universal structures** (independent of culture), and he proposed classifications of grammars known as the **Chomsky Hierarchy**. These formal developments resonated with the emergence of computer science.

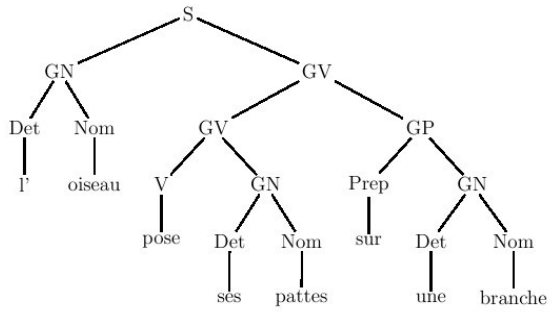


FIGURE 19 – Analysis through a syntax tree of the sentence: "l'oiseau pose ses pattes sur une branche" ("The bird puts its paws on a branch").

In 1965, the book titled *Aspects of the Theory of Syntax* by N. Chomsky introduced the concept of universal deep structures and peripheral structures. It presents language acquisition as innate development. An important concept is **recursion**, which allows sentence generation through the generation of **syntax trees** that exhibit a **hierarchy**. However, this is too **rigid** because, if we consider the syntax tree that analyzes the sentence "l'oiseau pose ses pattes sur une branche" ("The bird puts its paws on a branch") as shown in Figure 19, the words "pattes" and "sur" ("paws" and "on") that follow each other are "very far apart" in the tree. Therefore, this method has difficulty capturing **horizontal components of interactions**. Nevertheless, this theoretical foundation was central to the study of natural language: translation, interactions between speakers.

In the 1970s-1990s, **formal semantics** developed with tools like **mathematical logic** and **formal theoretical language** (λ-calculus being the first language defined and formalized for recursive functions). Richard Montague (1930-71) demonstrated in 1970 that **natural language could be treated as a formal language** (initially English, with generalization following). *Knowledge* can be captured by a set of *symbols* that will structure themselves into *propositions*. In computer science, **expert systems** and the **Prolog language** emerged. In 1985, Artificial Intelligence courses were based on these concepts and tools. When immersed in the field, it was very difficult to think that it could be something else. Indeed, if you are interested in what intelligence is, you quickly think of language, which involves linguistic structures, and you wonder how these structures are arranged. So, "it's obvious that it must be like this"![17] In the background of the developments of that time, the **Turing Machine** defined the framework that "simulated" the human mind. And, once the framework was defined, it needed to be filled.

3.3.2 Post-1990 View of the Problem

Now, let's take a **completely different perspective** that emerged around 1990 with Statistical Machine Learning and later with Neural Networks (NN). However, we must keep in mind the old notions because, as we will see, they do not entirely collapse.

The reason for this paradigm shift was the encounter with **the wall of complexity**. That is, as long as we are dealing with small problems, such as very simple sentences like "John loves Mary" or "John is going to the party with Mary," the old method works. But when multiple "actors" interact, there is an explosion of complexity that small, highly specialized linguistic corpora-based systems cannot handle.

A brief philosophical detour may help illustrate this point. We can consider two philosophical poles: **Empiricism and Rationalism**[18]. **Empiricism**, a more Anglo-Saxon notion proposed by Francis Bacon (1561-1626), John Locke (1632-1704), and David Hume (1711-1776), holds that **knowledge is based on the accumulation of observations and measurable facts, from which one can extract general laws through inductive reasoning, moving from the concrete to the abstract**. It opposes **rationalism (and innatism)**, which

17. NDJE: This is precisely what, in the history of science, is called a Kuhnian paradigm, named after Thomas Kuhn, a contemporary of Karl Popper, who described the conditions for scientific revolutions between "normal" phases where a "paradigm" prevails.

18. NDJE: This is a section I rewrote in consultation with philosophers. Of course, this is just a summary.

posits **discursive reason as the only possible source of all real knowledge**. **Rationalism**, as advocated by René Descartes (1556-1650), Gottfried Wilhelm Leibniz (1646-1716), and others, posits **the existence of universal logical principles** (principle of the excluded middle, principle of sufficient reason) and **a priori ideas**, i.e., independent of experience and **preceding all experience**.

While empiricism traces its roots to Aristotle's **"tabula rasa"** (the image of wax receiving impressions), it had significant extensions: analytical philosophy, logical empiricism of the Vienna Circle, and linguistics. Rationalism draws from **Plato, who exercises mathematics to detach us from the senses**, and from **Aristotle, who relies on concrete observation of nature (physis) to establish the foundations of formal logic (Organon), metaphysics, and ethics through reason**. All these approaches to knowledge can be found, among other places, in Physics and Mathematics. **Emmanuel Kant** (1724-1804) criticized Hume's empiricism because **for Kant, it is the subject that provides its rules to the object for knowing it** (there is a form of *a priori*). **He sets the limits of human understanding, which only has access to phenomena** (via experience), and **filters them through reason using an *a priori***. This is the Kantian synthesis.

Empiricism challenges causality: we only have access to successive events, and the causal relationship is only a relationship of temporal succession, with no guarantee that the same thing can happen again[19]. **Knowledge is not obtained through logical inference but rather through association**.

These two perspectives are at the heart of two completely different natures of mathematics. **The formal semantics before 1990 is more of a rationalist type (e.g., logic), while the ML/NN approach is more based on notions of analogy (geometry, distance)**. Whether we like it or not, we are subject to **cultural biases**, and while in France, we might say "tomorrow the weather will be nice or it will snow", in the USA, people might say "there is a 30% chance of it being nice or snowing," which is much more probabilistic. We will see how this manifests in the field we are interested in in this course.

19. NDJE: At the extreme, science is impossible

3.3.3 Machine Translation

In the 1990s, there was a shift towards a statistical approach with Markov chains related to speech analysis. However, gradually, there was a **divide** in the world of linguistics. On one side, there were people working on **theory** (structuralists, formal grammars), and on the other side, there were people working on **experience and statistics**, uncovering correlations that they used for sentence generation. The latter group was not well-received by the former…

Neural Networks belong to the second category. **In 2010, they achieved striking results**, and they form the basis of all current software for online translation (Google Translate, DeepL, etc.). There are cases where this kind of translator makes mistakes, but syntax and semantics are respected. However, a vast amount of data is still required for training. Interestingly, this architecture has been extended to **rare languages**! In practice, the most internal layers of the network are retained, learned from French and English, for example, and only the final layers are retrained with the target language. So, it's as if the **internal layers of the network capture *universal structures*** (similar to Chomsky's generic structures) independent of language.

In the end, we see the aforementioned divide disappearing. Because these **networks empirically (statistically) managed to capture language structures** (conceptual). Were these structures learned from training data, or were they already present from the beginning, i.e., in **the architecture of the network**, which is a kind of *a priori*? In a way, **"the innate element" is the network architecture (rationalism, innatism), and "the learned element" is the weights (empiricism, learning through experience)**.

In this empiricism/rationalism debate, the problem is therefore to understand to what extent the network architecture is fundamental (rationalism) or not, because then it would be learning that does the job (empiricism).

3.3.4 Describing a Video

Beyond applications focused solely on language, we are beginning to see **multimodal applications**: video and textual description, but more generally, sound-image/video-text-language. For example, consider the video https://vimeo.com/146492001, which provides an illustration of NeuralTalk2 from the Google Brain Team in 2016 (a network consisting of a CNN: convolutional NN followed by an RNN: recurrent NN).

3.4 Physics: N-Body Interactions

Let's recall that until very recently, Physics was the "only" science with a very high dimensionality and numerous interactions between particles, think, for example, of Astrophysics, Fluid Dynamics, and Quantum Chemistry. Over the centuries, we have been able to build a solid corpus of equations that govern these phenomena: Newton/Einstein, Boltzmann, Maxwell, Navier-Stokes, Schrödinger/Dirac, and so on. One could associate this discovery approach of particle interactions with a "rationalist" perspective.

One might think that, knowing these fundamental laws of dynamics, electromagnetism, and other forces between elementary particles, it would be enough to understand all of Physics because "all we have to do is work through the formalism". However, it is clear that the problem is extremely complex due to its dimensionality. **Fundamentally, all the fields mentioned (Astrophysics, Fluid Mechanics, Quantum Chemistry) involve N-body problems.** Simulations can help in some cases, but most of the time, many approximations are necessary!

However, "empirically", we see the emergence of **structures**: galaxies, two-dimensional turbulence, and molecules. So, instead of starting from the "simple" + equations to build complexity, can we not start from observations to capture complex structures? This can be formulated with the following question: **can we predict a solution through regression based on a database of solutions and some *a priori* information about the system under study?**

In fact, the answer is yes! And again, **with neural networks, this is being done better and better**. And **every time** (even for language), we see that the same type of network works, namely, **convolutional networks**. For example, in quantum mechanics, anti-symmetric wave functions of fermions seem to be captured by this type of network.

The applications are crucial because conducting, for example, **rapid simulations on complex systems** is at the heart of many industries: for example, the pharmaceutical industry for the manufacture of new molecules (e.g., calculating stability); materials physics; modeling dynamic flow around the profile of a car/plane. All these fields have been booming for the past three years. Note that these applications do not require predictions to be accurate to 10^{-12}, but they at least want to obtain the behaviors of these solutions.

3.5 Connection with Neurophysiology

Historically, around 1950, computational neurons emerged from biological models. Of course, the neurons in the brain are much more complex than formal neurons. Furthermore, in the brain, we now know that glial cells are at least as important as neurons. That said, in the functioning human brain, we see "specialized" areas (**large-scale structuring**), and we have identified neural plasticity (Marian Diamond, 1964), which explains the "reconfiguration" (**through learning**) of neurons that then take on the functions of deficient neurons.

What are the (mathematical) operations supported by these specialized areas? Note that if there is a connection between all these areas, we can think of the "elementary building blocks" of the brain. So, if CNNs work for "artificial" vision, it may be natural that they also work for "artificial" hearing because these two functions in the brain are supported by different areas, but fundamentally, these areas are composed of neurons (and other cells). And we can generalize this to language, memory, and so on.

In the last five years, there has been a lot of research on the connections between **neurophysiology and neural networks**. For example, the response of a neuron in the secondary visual cortex (back of the brain) to an input stimulus: can we build a mathematical model of this connection? Concerning the mathematical model, we have not succeeded, but can we build a neural network that performs the same function?

In fact, we cannot train such networks with physiological stimuli, so what is done is to take neurons from networks trained on images from ImageNet, for example, and by reducing the inputs, we try to see what the prediction capabilities of these artificial neurons are. What we find is that they are much superior to all elementary models obtained with linear auto-regressive models or with simple non-linearity, etc.

So, there is similarity between artificial neurons and physiological neurons. We also see this at the level of **invariants**. In 1981, groups of neurons in the visual cortex were discovered that are sensitive only to the recognition of simple bright bars on a dark background. Later, some neurons in the central nervous system were found to be specialized in detecting complex stimuli (the "grandmother cell" theory, which would recognize your grandmother's face). These functions have also been discovered in deep neural networks (see Google DeepMind in 2012).

Finally, we cannot avoid addressing **the subject in all its generality**. If at the mathematical level we juggle with geometric, statistical, probabilistic properties, we can see that this topic touches on many of the aforementioned fields, but we can also add social sciences and humanities because there is **a conception of knowledge behind it**.

3.6 Reinforcement Learning

We have seen that there is **supervised learning** and **unsupervised learning**. There is a third type called **reinforcement learning**. It's different because it is based on the **"trial and error"** principle. Instead of providing labeled examples (example + known response), we have a dynamic system with a feedback loop (reward if successful). The idea is to adapt the system gradually to optimize reinforcement (reward) to be as positive as possible.

One of the applications is learning for games, e.g., AlphaGo, which defeated the reigning world champion of the game of Go. In this case, it's important to distinguish in AlphaGo the convolutional neural networks (CNNs) (**note: there is local translation invariance and multi-scale structures**) that have been adapted for playing chess as well. There is an undeniable **form of creativity** compared to Deep Blue, which had beaten Kasparov. Deep Blue was based on a strategy of "gain" from a function with several depths of moves (utility function) by searching for the optimal solution in a decision tree. What constrained Deep Blue was the programming of the utility function that tells you "keep the queen or the bishop in such and such a situation". In contrast, **reinforcement learning does not use any of these *a priori* strategies**. But please note, the networks do not learn "from scratch" because there is still an underlying *a priori*: their architecture. However, apart from the architecture, we do not tell the network not to sacrifice the queen, for example. What we have seen is the emergence of new chess strategies (sacrifices) that have revolutionized the game.

Apart from the "playful" aspects, in the **industrial world, it is very important, for example, to train a robot to perform a particular task**: for example, the famous game for children of putting the right shapes in the right holes! By applying this type of

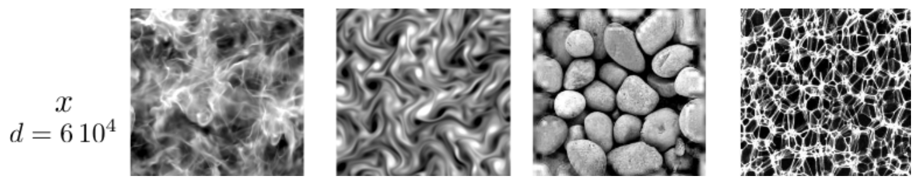


FIGURE 20 – Examples of different textures: a turbulent interstellar cloud, another type of fluid, a pile of stones, and bubbles.

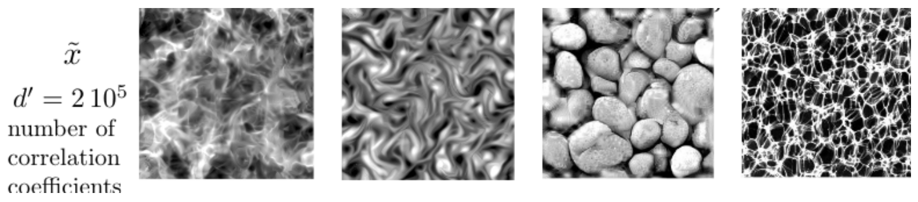


FIGURE 21 – Generation of new textures from those in Figure 20.

learning, robots can accomplish very complicated tasks through trial and error. See, for example: https://www.youtube.com/watch?v=JCjTQfy0h8w And there are many other videos showing groups of robots learning together...

3.7 Unsupervised Learning

Presumably, we won't have time to cover this type of learning this year but rather next year, however, it's important to keep it in mind. So, the idea is to establish **models from data**: for example, from data on fluid flows, we want to establish probability densities. Here too, spectacular results have been obtained using neural networks.

Let's take Figure 20 from left to right: a turbulent interstellar cloud, another type of fluid, a pile of stones, and bubbles. We take an image and input it into a convolutional network trained on ImageNet (animals, everyday objects, etc.) with no relation to turbulence. Inside the network, **we can calculate correlation coefficients between the sub-images produced by the neurons "stimulated" by this new image**. From these correlation coefficients, we can regenerate new images (Figure 21) from white noise that gradually reproduces the correlation matrices (**variational auto-encoder**)[20].

20. **NDJE**. Here, the network is trained with ImageNet, so the correlation coefficients do not contain

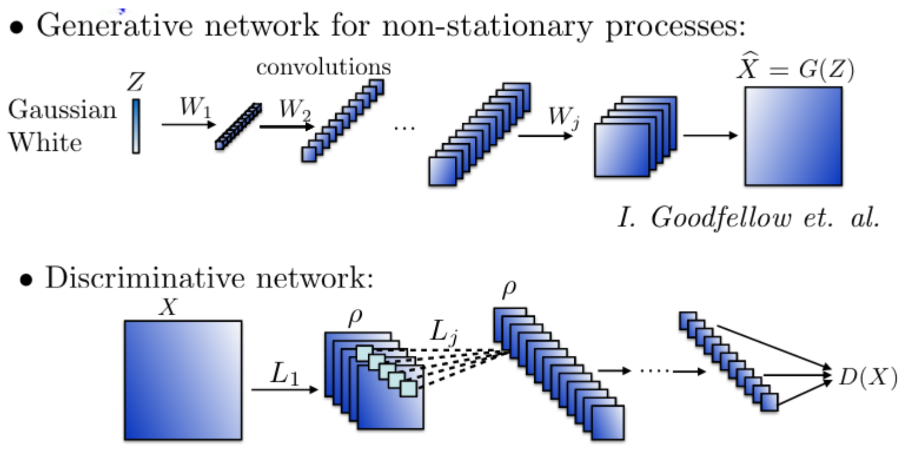


Figure 22 – Schematic architecture of Generative Adversarial Networks.

If we had determined correlation matrices of the images themselves to produce new images, we would have completely destroyed the structures[21]. So, how is it possible that this works? The nature of these correlations is an active area of research in statistics because **this defines new random processes with very sophisticated properties.**

3.8 Generative Adversarial Networks (GAN)

In the examples from the previous section, the images are **stationary**, meaning that local statistical properties are invariant under translation in the image. Can we do the same for **non-stationary** images? For example, if I take the image of one (or more) faces, can we **synthesize** new faces?

Six years ago, the answer would have been NO; we were not able to synthesize turbulence (stationary image) even though the Navier-Stokes equations date back to the mid-19th century, and Kolmogorov's theory dates to 1941. So, the community was particularly skeptical. The current answer is YES: Ian J. Goodfellow et al. in Yoshua Bengio's group in 2014. These are special convolutional networks called "generative autoencoders" that consist of two networks (Figure 22). It is clear that if we had only one network, giving it white noise would have no chance of working. **The two networks act in opposition**: the

any *a priori* information capturing the nature of turbulence images, piles of rocks, or sets of bubbles.

21. **NDJE**: think about what happens when we produce CMB maps from C_ℓ

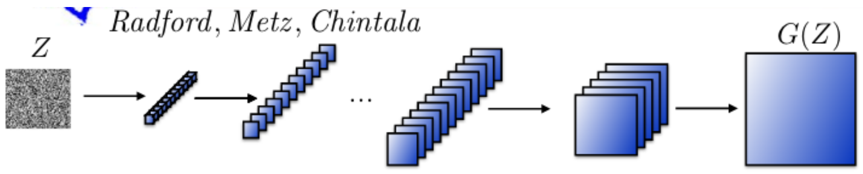


FIGURE 23 – Generation of a new image from white noise passed through a trained GAN.

generative network (G) produces new images, and the discriminator (D) provides the probability that the image presented to it comes from the database rather than a generated image.

We use a cost function that optimizes all the parameters of the two networks simultaneously:

$$\min_{\mathrm{G}} \max_{\mathrm{D}} \ \{\mathbb{E}[\log D(X)] + \mathbb{E}[\log(1 - D(G(Z)))]\} \tag{5}$$

This function states that the "discriminative" network should make as few mistakes as possible, but the "generative" network should deceive it as much as possible (it's the equivalent of a minimax two-player game). That is, D is trained to maximize the probability of correctly classifying an image as either from the database or generated by the model G, while G is trained to minimize $\mathbb{E}[\log(1 - D(G(Z)))]$ or, more practically, to maximize $\mathbb{E}[\log D(G(Z))]$. **At convergence, $D(X) = D(G(Z)) = 1/2$, meaning that the discriminator cannot distinguish between a database image and a generated image.**

Some examples: Let's take a GAN trained on images of bedrooms, so we present a white Gaussian vector z of about 100 parameters, represented here as a small image, but it can be a vector (Figure 23). It will generate an output image of 10^5 to 10^6 pixels $G(z)$. So, we present z_1, and G gives $G(z_1) = g_1$, then a new z_2, it produces $G(z_2) = g_2$. An interesting property is that if we make linear combinations of z_1 and z_2, the result is an image of a bedroom resulting from the mixture of the two images g_1 and g_2 (Figure 24).

We can do this kind of exercise with a database of faces (examples abound on the web), etc. Each time, we get a different G: $G_{bedroom}$, G_{face}, etc. However, **there is no information in the white noise at the input** (the z), so it must be in **the filters** of the G_i that there is coding (information) of what a bedroom is, a face, etc. **The generator has captured the important structures to reconstruct a face, a bedroom, etc**. If we look at it from a rationalist point of view, to put it briefly, we would break down a face into its

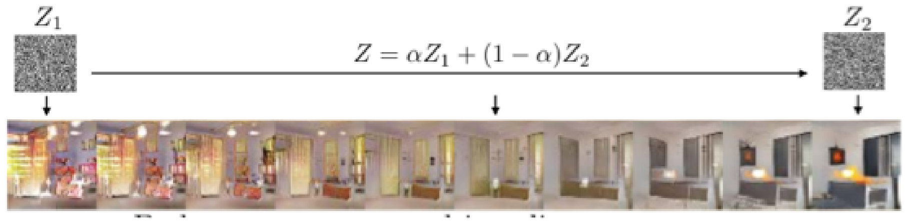


FIGURE 24 – Illustration of the property of linear combinations of GANs.

elements like the mouth, eyes, nose, and establish relationships (grammar) between them. **The underlying question: how is this structuring imprinted in the network?** Ultimately, what **type of memory, organization** is incorporated into these networks? We know how to program them, but currently, we are not able to describe how they work.

Note that there are applications in graphic art: painting a canvas in the style of a particular painter, or even interpolating artistic styles (see Gatys, Ecker, Bethdge 2015; https://arxiv.org/pdf/1508.06576.pdf). Let's not talk about generating "musical works", which leaves many of us intrigued, but it can be done.

3.9 Limits and Opportunities

3.9.1 The Dark Face to Illuminate

Currently, we are in an **essentially empirical phase**, with work focusing on **algorithms**. We **need vast amounts of labeled data**, and often it doesn't work due to a lack of training data. **What structures are encoded in the filters?** In fact, if we train the same network with different initial values, the weights will "individually" converge to different values, yet the network's performance is essentially the same! **So it's not the individual weights that matter but the entire network.** What is the regularity of the learned functions?

We **understand very little** about how a network works, what its performance, generalization capabilities, and limits are. Ultimately, **we have no *a priori* control over the result**. The only controls we can have are established from **statistical tests**. However, these tests are often **biased**. Indeed, let's revisit the conventional method that splits the training dataset into three parts: the **training set**, **cross-validation set**, and **test set**. Everything would go well if the training dataset is (was) representative of all the cases the network will have to deal with. However, there is a strong likelihood that the **training dataset is biased**, whether we want it or not.

Finally, the **architectures of networks** are most of the time **empirically optimized**, which is time-consuming and costly. We would like to have an "architect's guide".

3.9.2 The Motivating Face

The fact that the same type of architecture can yield excellent results in very different domains, i.e., the **generic** aspect of these networks, is indeed an enigma, but it contributes to the enthusiasm to understand **Why**, and to answer the question: what form of *knowledge* is being learned?

Regardless of these questions that some might call philosophical, the scientific prospects and applications are considerable. Indeed, if we are capable of learning to predict the evolution of **N-body dynamical systems**, then we become capable of conducting much finer **statistical physics** than that based on the study of disordered systems such as gases and crystals. The challenge is to perform physics at the **mesoscopic level** (chemistry, biology).

In mathematics, of course, there are many open questions, and in general, there are many research opportunities: and it's happening very quickly! Note that there aren't many people working on mathematics outside of algorithmics. Why? Because it's difficult, of course!

4. Mathematical Perspective

4.1 Introduction

Let's address, in this course (and the following ones), the questions raised by neural networks from a mathematician's point of view. We will focus on **supervised learning**.

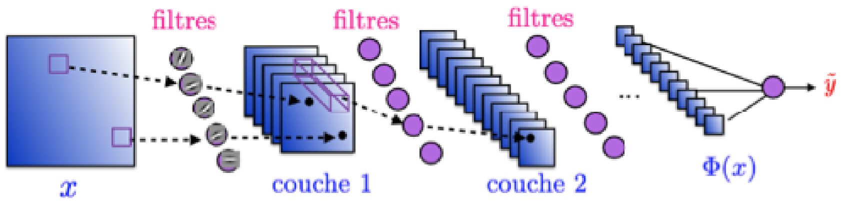


FIGURE 25 – Schéma d'un réseau convolutionnel CNN.

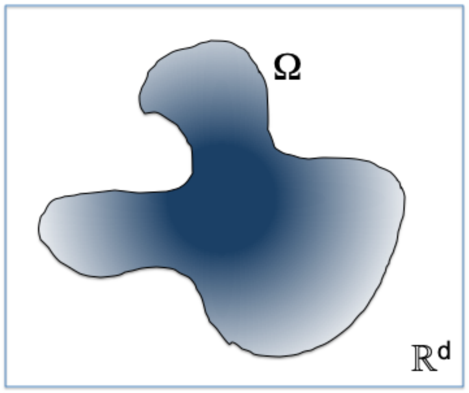


FIGURE 26 – Illustration of the probability density $p(x)$ of an image in the set Ω.

So, we will examine the structure depicted in Figure 25, where the input x (here a matrix) passes through the network, where the filters (W_i) are linear operators (matrices) associated with a non-linearity σ like a Rectified Linear Unit (ReLU), and by stacking them successively, we build the network architecture up to the last layer $\Phi(x)$, which is aggregated to give $\tilde{y} = \tilde{f}(x)$, the approximation of the function we want to compute.

4.2 Approximation Problem

Firstly, we have an **approximation problem**, which is the determination of W^*, a set of parameters (note: biases are also included) such that the approximation $\tilde{y} = f_{w^*}(x)$ is as good as possible. We define a notion of risk:

$$R(W) = \mathbb{E}_x(r(f(x), f_w(x))) \tag{6}$$

Here, $r(y, \tilde{y})$ is a penalty function for making an error. Typically, for Regression and Classification problems, we use the following risks (non-exhaustive list):

Regression	Classification
$y \in \mathbb{R}$	$y \in \mathcal{A}$
$(y - \tilde{y})^2$	1 if $y \neq \tilde{y}$, 0 otherwise

Ideally, the solution W^* minimizes the risk *on average*, which means:

$$W^* = \underset{W}{\operatorname{argmin}}\ R(W) \tag{7}$$

This assumes that we know the probability distribution $p(x)$ of the data x over Ω. For instance, in Figure 26, darker colors represent higher probability density. Under what conditions is the error $R(W^*)$ small? In other words, we would like to ensure that for a given ε:

$$R(W^*) \leq \varepsilon \tag{8}$$

However, in high dimensions, this problem becomes challenging. When making classical assumptions about the function f having a certain degree of regularity on $x \in \Omega$, such as being Lipschitz (with almost everywhere bounded derivatives), achieving a small error requires having a sufficient number of examples that cover Ω well for interpolation of f (see Figure 27). How many examples are needed? Typically:

$$n \approx \varepsilon^{-d} \tag{9}$$

The same scaling applies to the number of parameters needed to fit these examples:

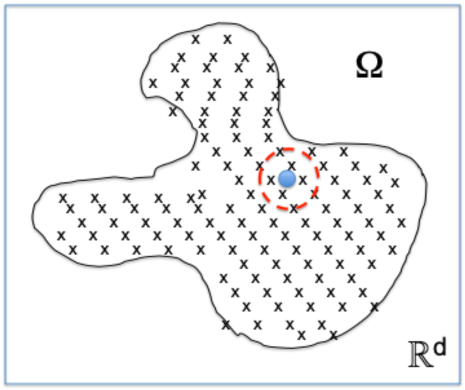


FIGURE 27 – Illustration of the interpolation problem in high dimensions. There are few or no known samples close to a new sample.

$$W \approx \varepsilon^{-d} \tag{10}$$

However, d is very large, leading to what is known as the "curse of dimensionality". Therefore, strong regularity assumptions beyond Lipschitz are needed to achieve a small error, indicating that the data points x in Ω are highly structured. What are these new forms of regularity?

This is a field of mathematics that is well understood when d is small ($d \leq 3$), but in high dimensions, it's a different story, which we will explore later. Before that, let's address another problem.

4.3 Estimation and Optimization Problem

In the previous section, we assumed that we had the "best solution" and asked if this solution met a quality criterion (minimum cost/risk). Now, we need to find this solution W^*. Why is this difficult?

Firstly, we need to calculate the expectation $R(W)$:

$$R(W) = \mathbb{E}_x(r(f(x), f_w(x))) \tag{11}$$
$$= \int_{\mathbb{R}^d} r(f(x), f_w(x))\ p(x)\ dx \tag{12}$$

However, $p(x)$ is completely unknown because we only have information about it through a limited set of training examples.

So, what we can do is calculate an estimate of the risk $\tilde{R}(W)$. We move into the domain of empirical statistics. If we have constructed a neural network, we can know the empirical mean risk based on the examples used:

$$\tilde{R}(W) = \frac{1}{n}\sum_i r(y_i, f_w(x_i)) \tag{13}$$

We would like the behavior of this estimate as n approaches infinity to be such that:

$$\tilde{R}(W) \underset{n\to\infty}{\longrightarrow} R(W) \tag{14}$$

Under what conditions is this true? We need to control the variability of the empirical estimator; for example, we need to control the maximum error:

$$\max_W |R(W) - \tilde{R}(W)| \tag{15}$$

However, this will depend on the number of parameters in W. Ultimately, we want:

$$\tilde{R}(W^*) = \min_W \tilde{R}(W) \tag{16}$$

This means that if we feed an example x_i into the "optimized" neural network with parameters W^*, the difference between y_i and $f_{w^*}(x_i)$ should be as small as possible.

The more parameters we want to adjust, the more examples are needed to keep the error small. This leads to a similar problem as before. Typically, $n \sim O(10^{5-6})$ while the number of parameters is often $O(10^8)$.

However, if it works in practice, it's because of a **form of regularization** that occurs. The network will adapt to adjust only the most relevant parameters for providing the answer.

This notion of **regularization** is the other main theme of neural networks that we will explore, and it is often *implicit*. Even without explicitly imposing regularization, it occurs. It's crucial to understand the nature of this regularization.

Now, we also need to **find the solution** $\tilde{W}^*$ that minimizes the empirical cost. The well-known mathematical framework for this was convex function minimization, where simple gradient descent methods work perfectly. The problem with neural networks is that we are not in this case at all. We have a plethora of local minima. So, a priori, gradient descent methods will not find the right solution. However, surprisingly, it works. We often end up in local minima, but the "distance" to the global minimum that we estimate is usually small: 1) **if the network is well-configured**, 2) **if the right gradient descent method is chosen**, and 3) **if the algorithm's parameters are tuned correctly**.

This is an empirical field: **non-convex optimization** attempts to obtain theorems that provide guidance for creating a good network. **This optimization will be central in this year's course** because, at the very least, even if we are not interested in the mathematics, we would like to have ideas for designing a network, knowing how to train it, and what to do if it doesn't work.

The set of network parameters W consists of the set of matrices W_i and the biases of the non-linear functions, so for J layers:

$$W = \{W_i, b_i\}_{i \leq J} \tag{17}$$

The number of parameters is quite substantial, numbering in the millions.

For each set of parameters W, we obtain a different $\tilde{f}$, so it is natural to use the notation f_w. Therefore, by varying W, we get a whole class of functions f_w, denoted as $\mathcal{H}_w$:

$$\mathcal{H}_w = \{f_w \ / \ \forall \ W\} \tag{18}$$

$\mathcal{H}_w$ is the set of functions that can potentially be programmed with a neural network.

In all the examples we have seen in the previous chapters, what we are looking for is

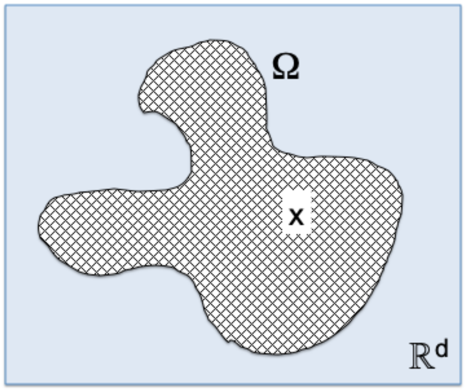


FIGURE 28 – Structured images are part of a set $\Omega \subset \mathbb{R}^d$.

to find an approximation of f such that $y = f(x)$, with $x \in \Omega$, where Ω is, for example, a set of structured images, a subset of all possible images of the same dimension. We know that $\Omega \subset \mathbb{R}^d$ with d being the dimensionality of the problem (Figure 28).

We don't know f or, *a fortiori*, the (W_i, b_i), and we don't know the topology of Ω any better… but in supervised learning, we have examples: $\{x_i, y_i = f(x_i)\}_{i \leq n}$. So, we want to approximate $\tilde{y}$ with a function from a specific neural network, i.e., $f_w = \tilde{y}$.

With this general framework set, how do we get the right neural network, or how do we determine the W_i and b_i? Recall that we have an enormous class of possible functions (cf. $\mathcal{H}_w$), so what is the right strategy? In fact, we have two sub-problems.

4.4 Other Questions

Returning to the approximation problem, we ask **how and why do these network architectures**, even if we manage to optimise them, **give a good answer**? We try to understand what functions $f(x)$ can be approximated by a network. For example, if you start with a quantum N-body chemistry problem that must satisfy a potentially very complicated Schrödinger equation, why can we ultimately approximate the solution with a network with 10^8 parameters and very few examples? But more generally, **why are images, sound, language structured, or what structures are we talking about?** This type of question arises when approaching the approximation problem in the form of: what is the **notion of regularity**? Another question is **how are all these problems similar?** Is there a **common notion of regularity**?

4.5 Approximation/Regularity, What Type of Regularity?

As we have seen several times, if we only have **local** regularity (Lipschitz), **it is not sufficient**, as we are not able to provide enough samples in the vicinity of a particular x to infer the value of $f(x)$. So, we need notions of **global** regularities that invoke **symmetries**. We will introduce an **operator** (in the manner of physicists...) g that will move x to x':

$$x \to g.x = x' \tag{19}$$

How will the function f react to this transformation? That is, instead of looking at $|f(x) - f(x')|$ with x' in the vicinity of x, here we look at $f(g.x)$. Now, **if g is a symmetry of f, then**

$$\forall x \in \Omega, f(g.x) = f(x) \tag{20}$$

Note that $|g.x - x|$ does not need to be small, unlike Lipschitz regularity.

Can we describe the properties of f based on its symmetries? Intuitively, the more symmetries a function has, the more regular it is. A quick reminder: E. Galois introduced the **symmetries of the roots of algebraic equations** of degree d to show that they cannot be expressed in radical form when $d \geq 5$; that is, he was able to understand the nature of the solutions.

If g_1 and g_2 are 2 symmetries of f, then

$$f(g_2.(g_1.x)) = f(g_1.x) = f(x) \tag{21}$$

meaning that $g_2.g_1$ **is also a symmetry of f**. Therefore, we can study the set of symmetries of f:

$$G_f = \{g/g \text{ symmetry of} f\} \tag{22}$$

which has a **group** structure with basic properties:

— The composition of 2 elements is also an element of the group.
— The existence of a neutral element.
— The existence of an inverse for each element.
— The associative law of composition.

4.6 Emergence of Symmetries and Local Groups

4.6.1 From Global to Local

Let's see in the practical case of image classification, what kind of symmetry is at play and how does knowing these symmetries help us understand the problem? An image x is composed of pixels that we denote by u, $x(u)$ is the pixel value. If we perform a **translation**:

$$x(u) \to g.x(u) = x(u - g) \tag{23}$$

the nature of the object in the image does not change, which implies that:

$$f(g.x) = f(x - g) = f(x) \tag{24}$$

Also, if we perform a **rotation**:

$$x(u) \to g.x(u) = x(R_g.u) \tag{25}$$

In some cases, this won't change the problem (e.g., recognizing a dog, galaxies), but in some cases, it can pose recognition problems (e.g., sky/sea, or the digit 6 vs. 9). **So, rotational invariance depends on the problem**.

Now, how does knowing the type of symmetry help us? In fact, **we can reduce the dimension of the problem**. Initially, $x \in \Omega$ has a huge dimension d; however, if there exists a symmetry group G, we can **reduce (by quotienting) the study space to the set** Ω/G (think of arithmetic, modulo $\mathbb{Z}/p\mathbb{Z}$, which allows us to reduce the study to remainders after division by p).

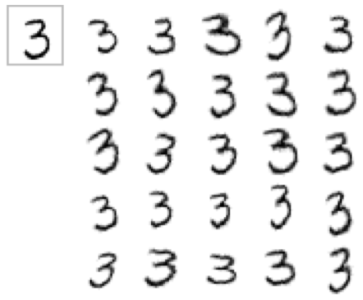


FIGURE 29 – Examples of deformations of the digit 3.

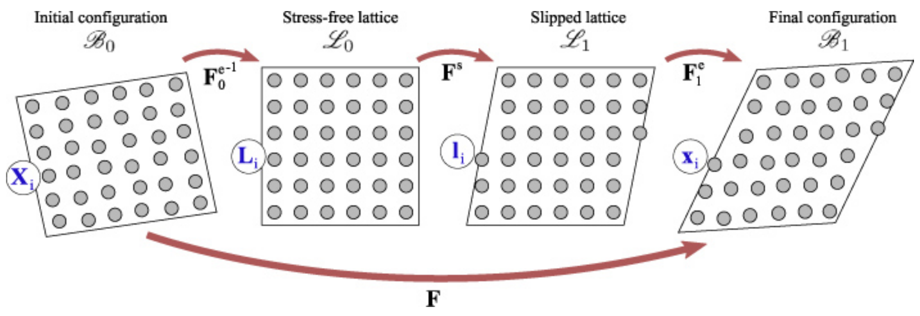


FIGURE 30 – Different types of deformations that can be applied to an image.

For translation, we can subtract 2 parameters (for a 2D image), but if d is very large, $d - 2$ is still very large! So, if we want to significantly reduce the dimension of the problem, we need to obtain **very large symmetry groups**. What can we think of? In the analysis of digits, for example, we can consider groups of **deformations**. For example, in Figure 29, there are different distortions of the digit 3.

We can formalize the result of a **deformation with "local translation"**:

$$x(u) \to g.x(u) = x(u - g(u)) \tag{26}$$

(it's a local group). Note that we always use the same notation $g.x(u)$ to highlight the concept of transformation; otherwise, we can get into details, as shown in Figure 30, but that would make the notation cumbersome.

We can then see that instead of **2 parameters to define a global translation**, now the group G of **local translation** will be indexed by **2 regular functions**. Therefore, the

dimension of G can be very large. In the end, if this type of symmetry is present in the problem, the dimension d is greatly reduced. We have gone from **discrete groups** to **Lie groups** of Marius Sophus Lie (1842-1899), well-known in Theoretical Physics, which describe the underlying regularities of functions $f(x)$.

However, there is something to keep in mind, and that is that there is an element of **low dimension behind it** that allows us to tackle the problem, namely, **the dimension of the space in which u evolves**. In the case of a 2D image, u indexes the position within the image, so we have 2 independent parameters; for sound, it's time that indexes, which is 1 parameter; similarly for text, there is an ordering of letters... even in Physics, the underlying dimensionality is only 4 dimensions (unless otherwise). And this is **absolutely fundamental** because in all convolutional networks, convolutions act on the variable u. Discovering the symmetries of the problem means applying transformations to the variable u and observing how f behaves.

Let's look at **audio**. It is natural to study frequencies and harmonics (as seen in the spectrogram). If there are spectral regularities, we want to perform **frequency translations**, which are modeled by:

$$x(u) \to g.x(u) = x(u)e^{ig.u} \tag{27}$$

But here too, instead of taking a **global transformation**, we can consider **local transformations** of the form:

$$x(u) \to g.x(u) = x(u)e^{ig(u).u} \tag{28}$$

In practice, we might then ask: can we recognize a sound even if it is distorted by a vocalization produced by a singer? If we want to tackle this problem, then the function must be invariant to local deformation in the frequency space.

What we observe through this example is that by knowing the space in which u operates and the type of problem posed, we have access to different types of symmetries. However, **how do neural networks capture the symmetries of the problem**, which belong to very high-dimensional groups? Moreover, we do not fully understand the subject of **deformation groups in high dimensions**, which is the study of **diffeomorphisms**, one example of which is shown in Figure 31 for a 2D square.

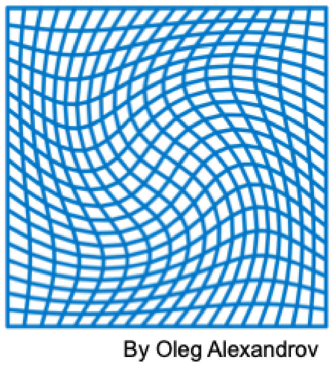


FIGURE 31 – Example of a diffeomorphism applied to a square.

4.6.2 Impact on the Neural Network

A network performs successive transformations:

$$x \to \Phi_w(x) \to f_w(x) = \sigma(\langle \omega, \Phi_w(x) \rangle) \tag{29}$$

with a linear transformation from $\Phi_w(x)$ to $f_w(x)$, followed by a non-linearity σ. So, if we know that the problem has the symmetry g, then:

$$\Phi_w(g.x) = \Phi_w(x) \Rightarrow f_w(g.x) = f_w(x) \tag{30}$$

We say that g is a symmetry of f (and of Φ), which is equivalent to saying that f (and Φ) is invariant under the action of g.

Thus, gradually, during training, the network constructs a representation Φ that has the right symmetry. **Experimentally, we observe these invariants in networks**. Moreover, we also observe them in the networks of the brain in the upper cortical areas (e.g., for sound, vision). And in the case of the brain, we observe much more subtle symmetries because we can recognize a face in many conditions that could be described as "degraded".

So, we always come back to the same question: what types of invariants are learned by the network? Because if we can design networks with *a priori* symmetry properties (e.g., translational symmetry $\Rightarrow$ convolutional filter), how are they also capable of learning other types of symmetry? and what do they learn? We know that it's not the parameters

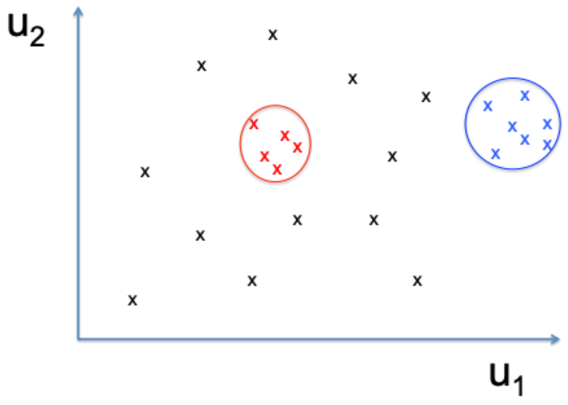


FIGURE 32 – Illustration of *short-distance* interactions within a group.

themselves that carry meaning; there are invariants that make two solutions obtained by two different trainings equivalent. The study of symmetries can attempt to understand these invariants.

4.7 Scale Separation, Multi-Scale Hierarchy

When we have **many variables**, such as pixels in an image, letters in a text, atoms in physics, agents in a social network, and it is **the interactions** between these variables that are relevant to solving the studied problem, what we observe is that the **actors who interact the most are those who are closest to each other** (Figure 32).

In a social network, a person will interact with their family and friends; in an image, a pixel will interact with its neighbors, such as the red crosses in Figure 32. However, potentially, one must also take into account **long-distance interactions** (between red and blue crosses in Figure 33), even if they are of much lower amplitude: in a social network, should we not consider the influence of a distant country to understand the voting behavior of people nearby? In an image, we know that the background can also have large-scale components; in statistical physics, we also know the importance of long-distance forces in spin glasses, etc. Why should we not neglect these long-distance interactions? The reason is that, as we have many variables (e.g., several billion in a social network, etc.), by neglecting interactions with many agents, we neglect the sum of these interactions compared to local

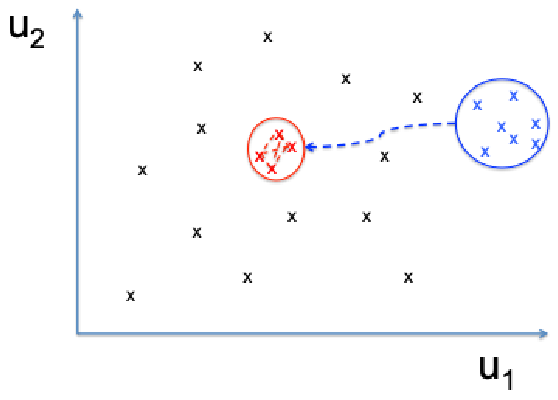


FIGURE 33 – Illustration of *long-distance* interactions between groups.

interactions, **yet they can potentially be of the same order of magnitude** due to the high number of interactions between agents at long distances. However, while at short distances, we can take into account all interactions between closer neighbors, the more distant groups are considered, the more we can consider initially only an "averaged" interaction.

Therefore, through this **averaging operation**, which takes more and more variables (agents, particles, pixels, etc.) as they become more distant, **we transition from a problem in dimension** d (the number of agents, particles, pixels, etc.) **to a problem in dimension** $O(\log d)$. This is a phenomenon of **hierarchical**, **multi-scale** processing, which we will come back to and which is very intuitive (it even has roots in Descartes' "Discourse on the Method": "Divide each of the difficulties I examine into as many parts as possible and as is required to resolve them better"). This is very important because potentially **it allows us to overcome the curse of dimensionality**.

This is manifested in convolutional neural networks (CNNs) like the one in Figure 34: as we progress from the input to the deep layers, there is a reduction in information (convolution and sub-sampling layers), but it becomes increasingly complex and aggregates pixel ranges from the initial image.

Thus, **this hierarchy of scales allows for a massive reduction in the number of network parameters**. However, the problem is still challenging because, while we can consider "averaging" interactions as we consider larger scales, large groups of agents (pixels, etc.) far from a region of interest also have interactions among themselves, as illustrated in

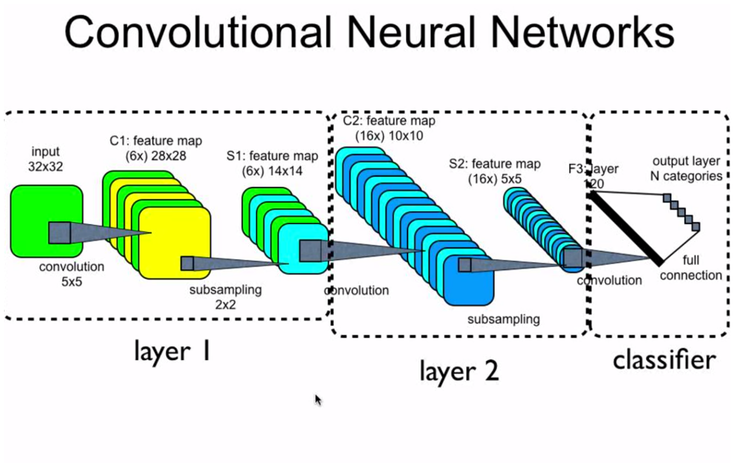

Convolutional Neural Networks
input 32x32
C1: feature map (6x) 28x28
S1: feature map (6x) 14x14
C2: feature map (16x) 10x10
S2: feature map (16x) 5x5
F3: layer 120
output layer N categories
convolution 5x5
subsampling 2x2
convolution
subsampling
convolution
full connection
layer 1
layer 2
classifier

FIGURE 34 – Detail of convolution and sub-sampling operations in a CNN that reveal interactions at various distances between pixels of the initial image.

Figure 35. For example, we can consider a geopolitical problem where one's own actions (e.g., purchases) are influenced by distant countries whose behavior we would like to understand on a country-by-country basis, but perhaps it is the tensions between these "distant" countries that are relevant to our affairs.

What mathematical tool will be used to capture this hierarchy of information? In the 1980s-90s, there was a tool: **Wavelet Theory**[22]. Note that the first idea that comes to mind for hierarchical organization is that of a **tree structure**: for example, a company broken down into divisions, services, with humans at the end; we also saw these trees in Chomsky's grammars. In this type of formalization, we don't really need mathematics, although we can develop algorithms such as *decision trees*. But this relatively simple and standard tool **doesn't work** in our problem because, even in a company, we know very well that two office neighbors who don't speak to each other can be far away when we go up the hierarchical chain. The problem also arises concretely for Chomsky grammars when scaling up. **We want to introduce a horizontal structure at all "nodes"** so that we can exchange information, or in other words, to make the structure more flexible. **Random Forests** are decision trees, but **there are many of them**, and we average them to account for both hierarchical structure and variance at all scales.

22. NDJE: S. Mallat briefly mentioned it in 2018, and I provided a fairly detailed appendix on it.

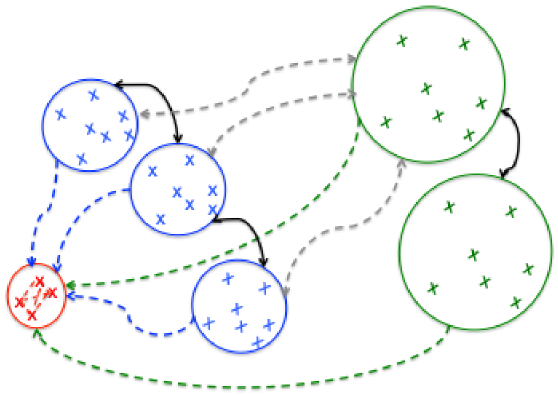


FIGURE 35 – Illustration of the role of interactions between groups at different scales.

In convolutional networks, information "naturally" communicates in the filters. **How these communications are related to the notion of symmetry is a subject of study?**

4.8 The Notion of Sparsity

The notion of "sparsity" actually applies to all learning algorithms and is generally used in low dimensions. It is often associated with **pattern recognition**, as if there were some **basic elementary patterns** that needed to be recognized to understand the problem. Intuitively, this makes sense: in a face, there are eyes, a nose, a mouth; in a text, there are words, phrases, paragraphs, etc. So, we want to highlight **patterns** that structure the entirety of a face, text, etc. However, depending on the problem, there can be many elementary shapes that we probably want to organize into **families**. Mathematically, the idea of decomposing a function into specific families falls under the realm of **Approximation Theory**. We then define a **basis** $\mathcal{B}$ that includes a very large set of orthogonal vectors ($N \sim e^d$):

$$\mathcal{B} = \{g_n\}_{n \leq N} \tag{31}$$

Thus, any function can be decomposed as:

$$f(x) = \sum_{k=1}^{N} \alpha_k \; g_k(x) \tag{32}$$

However, it is wise to reveal those **patterns/features** that are not necessarily orthogonal because they may be redundant but better represent the function f. So, we take a subset of coefficients that still provide a good approximation:

$$f(x) \approx \sum_{k \in I} \alpha_k \; g_k(x) \tag{33}$$

By taking only a subset of coefficients, this means that the others are set to 0. Therefore, finding a **sparse** representation means finding a representation (basis) in which **most of the coefficients are zero**, and the few **non-zero coefficients represent the elementary patterns**.

A natural tool that comes to mind in this context is **the Fourier basis** $e^{im.x}$ in which the function f is decomposed. Indeed, if **the function** f **is very smooth** (in a **Sobolev space**), then the low frequencies are well representative, while **high frequencies** will be scarce or **non-existent**. However, if there are **discontinuities**, we will need to adapt the basis.

In general, we need to find **dictionaries** (not necessarily linearly independent, any pattern is accepted):

$$\mathcal{D} = \{g_n\}_n \tag{34}$$

The representation is said to be **sparse** if we can approximate the function f by f_M with a limited number M of coefficients:

$$f_M(x) = \sum_{k \in I_M} \alpha_k \; g_k(x) \quad \text{and, } |\text{card}(I_M)| = M \tag{35}$$

We would like the approximation to converge quickly to the function as we increase M, i.e.,

$$||f - f_M|| \leq C M^{-\alpha} \tag{36}$$

Sparsity is therefore about finding a basis, a family, or a dictionary that allows this type of approximation by reducing the number of coefficients.

This type of scheme **can be implemented in low dimensions**. Here, we refer to the dimensionality of u. For example, in an image x, we index the pixels with 2 parameters if we want to maintain the row-column structure (u_1, u_2), and the pixel's value is $x(u)$.

Thus, in the case of an image (similar for sound), efficient compression algorithms have been developed. But in the case of a function f **with a very large number of parameters**, for example $f(x(u))$ as a function of x, then **the number of elements in the dictionary will grow exponentially**.

In this case, **we will need to use different tools from the outset**. We feel that there is indeed an aspect of very high dimensionality in certain problems (e.g., chemistry, physics, etc.), but in a way, if we work with molecules of a certain type and know their interactions, we can forget about the quantum aspect and Schrödinger's equation. These are concepts at work in classical approximation theory. The question then is: **how and where do these structures appear in neural networks**? In recent publications on neural networks, authors try **to reveal these patterns in filters**. For example, in digit recognition, a certain filter may capture a specific shape, corner, and so on.

So, in a first approach to a problem, it becomes natural to question patterns. **But this is not enough**. In fact, we cannot capture very complex phenomena in high dimensions that give rise to new properties/structures. Think about the textures of wood, stone, fabric, etc. If we want to account for all the different types of textures we encounter in nature, there are countless variations. Yet, humans easily perceive different textures in their environment.

Therefore, it is likely necessary to have a global perspective. This translates to **understanding the structure of the entire set** Ω in which x evolves. Typically, this is what is examined in unsupervised learning. For example, if we work with organic molecules (Ω) and begin to understand how they interact, then train a network with these molecules, it is clear that if we provide it with a molecule of a different type (ionic crystal), it may not work well *a priori*! In conclusion, **the network will become highly specialized in the elements of** Ω.

Here, you can clearly see the difference between this type of learning and one that involves understanding the Schrödinger equation or another equation in physics that works outside of Ω. Ultimately, even in very high dimensions, by restricting ourselves to Ω, we simplify the problem (to some extent).

4.9 Summary from a Mathematical Perspective

What has been described in the previous sections is more of a research program. In the following sections, we will address the algorithmic, optimization, and estimation aspects. When studying a network with 1 hidden layer (often called 2-layer networks), the theory of approximation is well understood. However, beyond that, there is an explosion of complexity, and we have a much harder time understanding what is happening.

5. Where Do the Ideas of Neural Networks Come From?

5.1 Cybernetics

The history begins with the American mathematician **Norbert Wiener** (1894-1964), whose ideas are presented in his book *Cybernetics or Control and Communication in the Animal and the Machine* (1948), which had a significant impact beyond the scientific world. Wiener's father was a Russian immigrant to the United States and had a vision of the child as a completely malleable clay, so he had a project to turn his son into a genius. Along with a friend, also a Russian immigrant who shared his educational views, they traveled across Europe to provide their children with a historical and literary education, while immersing them in science at home. It turns out that Norbert developed a prodigious memory due to vision problems that made him focus on his hearing. And, in the end, both Norbert and his friend became child prodigies. Both children earned their Ph.D. degrees around the age of 18 at Harvard, with Norbert specializing in mathematical logic. However, his friend couldn't handle the pressure and ended up committing suicide.

Norbert, on the other hand, created and revolutionized several fields, including signal processing (e.g., the eponymous filtering), control theory (e.g., he introduced feedback loops, motivated by anti-aircraft gun tracking of airplanes), Brownian motion has a "Wiener measure", and he revisited Fourier analysis, among others. So, his extraordinary creativity extended far beyond the realm of science.

According to Wiener, **cybernetics is a "complete theory of control and communication, whether in the animal or in the machine"**. In this framework, **learning is seen as**

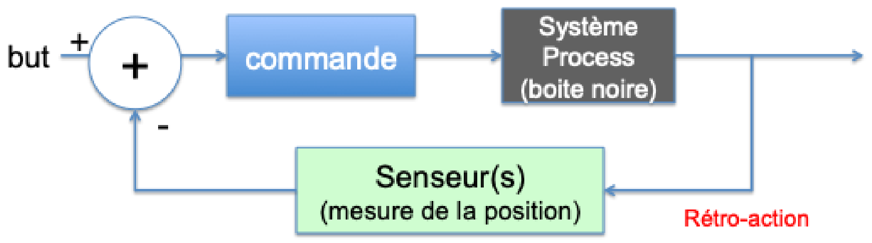


FIGURE 36 – The different phases of a control loop to reach port.

a dynamic system. **Intelligence** is viewed as **an adaptation to reality**, and this adaptation cannot be thought of without **considering time**. Thus, there is a system that evolves over time and must adapt. One consequence is that **we do not model the world but rather the way to react to the outside**. A classic example is that of a "boat returning to port" with external conditions such as wind, currents, and waves. Two approaches are possible: **either we model the entire external environment and the operation of the boat itself, or we focus solely on the boat's course and speed** to achieve the objective. The figure 36 illustrates the different phases of a control loop.

This very general system is that of **negative feedback**, where one acts based on **error**. This idea has been extensively developed not only in engineering but also in social science and biology. This led to the development of a **cognitive robot project**, the (caricatural) functioning of which is schematically represented in figure 37. The "goal" or "objective" achieved is an input to the "Planner" part of the cognitive system (a primitive version of the brain), which also includes a "World Representation" and a "Perception" module to interact with it. People discussed this scheme extensively, trying to figure out how it could be implemented in computer (and mechanical) systems. This was fundamental for the field of **"control systems"**.

Developments in cybernetics occurred in the 1940s to 1960s. There was a lot of analysis of the scheme (figure 36) in relation to what could be observed in nature. An article by **Herbert Simon** titled "***The Architecture of Complexity***" (1962) highlighted that learning is effective when **the world is structured** and not too complex. Complexity, for example, determines the number of learning samples, but in general, it sets a limit on

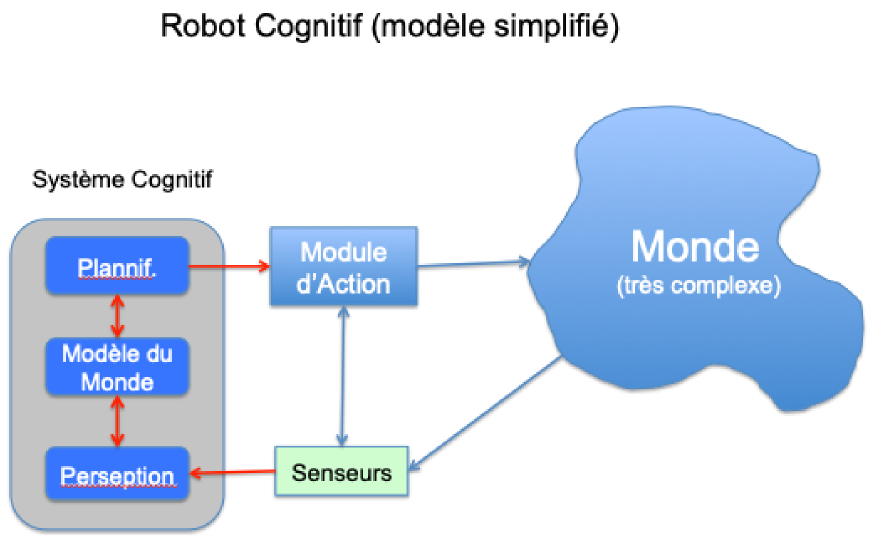


FIGURE 37 – The "Cognitive Robot" project.

how much of the world can be learned. H. Simon also showed that there is a notion of **ubiquitous hierarchy** observed (*reductionism*) in Physics (particle, molecule, atom...), in Biology (cell, tissue, organ...), in the symbolic domain (e.g., language: letter, phoneme, word, sentence, paragraph, chapter...), in History with the evolution of states (tribes, village, town, region, state, empire...), and generally in solving mathematical problems (hierarchy of theorems...).

One of the questions that emerged is: *Why do we observe this hierarchical structuring everywhere?* H. Simon's thesis about the existence of these hierarchies is the need to have something that is **stable** while being **adaptive over time**. To illustrate his point, he used the image of a watchmaker who assembles a very complex watch during a step where he holds all the pieces together but is continually interrupted by customer calls. The result is that this poor watchmaker will have to restart his assembly very, very often, or he may never succeed in assembling the watch. The watchmaker will probably change his operating mode to assemble pieces and then assemble larger and larger subsets and finally assemble the "big pieces" together. If he is interrupted, he will have less work to redo and will converge toward assembling the watch. This image represents Darwinian **evolution**, which structures elements hierarchically, from stable elements to all scales. **H. Simon**

describes complexity through the hierarchy of structures, and this hierarchy allows us to understand complexity. This idea is crucial because it lies at the heart of all **deep neural networks** in which "depth" introduces **notions of structure and hierarchy**.

The consequence of hierarchy is that there is an organization into **subsystems** that are **quasi-separable** and therefore weakly linked. This qualitative idea is quite natural and is not new. However, this approach has been a failure! The real question is: how to represent weak interactions without eliminating them (see section 4.7). How to represent these hierarchies, how to represent the different states at all scales while integrating the notion of sparsity so that we can "learn this structure". This is the question that spans all the disciplinary fields mentioned above and one to which the community has not been able to answer for a long time, as evidenced by Chomsky's grammars. It is also at the core of the understanding of how deep neural networks (DNNs) operate.

So, the underlying ideas of DNNs are not new, we are at the "algorithmic" stage, meaning we can implement them and make them learn structures, but "mathematically" we do not understand how they work.

5.2 The Perceptron (1957)

5.2.1 Introduction

This is the idea of **Frank Rosenblatt**, an American psychologist who invented a single-layer learning network[23]. It's worth noting that Wiener, Chomsky, Rosenblatt were all at MIT, as were Marvin Lee Minsky and John McCarthy, who founded the Artificial Intelligence Group. Minsky and McCarthy developed the logical and symbolic aspects.

The problem posed is a classification $x \in \mathbb{R}^d$

$$f(x) = \begin{cases} -1 \\ \\ 1 \end{cases} \tag{37}$$

23. Note: In 1943, W. Pitts and W. McCulloch introduced a "formal neuron" capable of performing logical functions, four years before the realization of the first transistor.

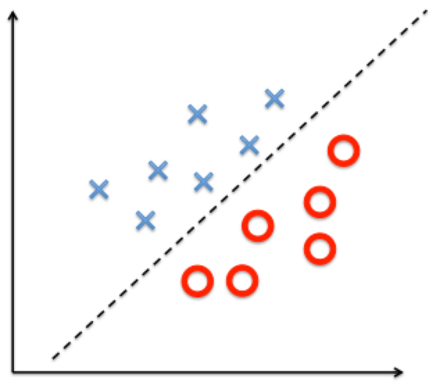


FIGURE 38 – Example of 2-dimensional classification.

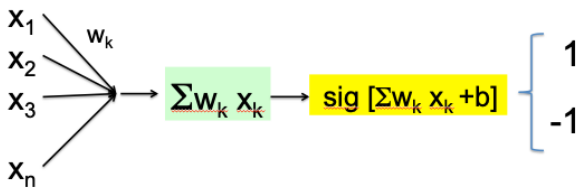


FIGURE 39 – Graphical representation of what a linear classifier should do.

and the goal is to separate sets of x_i by finding a "boundary" between the two populations, as shown in figure 38. The equation of the boundary is given by

$$\langle w, x \rangle + b = 0 \quad \text{and we choose} \quad ||w|| = 1 \tag{38}$$

where w is the orthogonal direction and b is the distance from the origin of the axes to the line. The "x"s belong to class 1, and the "o"s belong to class -1. Given the class y_i of all x_i in the training sample, we want the x_i to be on the "correct side" of the boundary so that

$$y_i(\langle w, x_i \rangle + b) > 0 \tag{39}$$

This can be graphically represented as shown in figure 39, where the non-linearity is the "sign" function.

The question is to find the components of the vector w (i.e., w_k) to achieve the separation of the two classes with an iterative algorithm that will update the weights to converge to the solution. The basic ideas of neural networks were defined at this very simple stage.

The first idea is to introduce **a cost (risk) for misclassification**. So, we will take all the misclassified points $i \in \mathcal{M}$, and the Rosenblatt risk is given by

$$\boxed{R(w,b) = \sum_{i \in \mathcal{M}} (-y_i)\ (\langle w, x_i \rangle + b)} \tag{40}$$

In the above expression, "$-y_i$" gives the "wrong class", and "$\langle w, x_i \rangle + b$" gives the signed distance of the point to the line; it is the penalty for misclassification. In other words, the further the point is from the correct side of the boundary, the greater the penalty.

The second question is to find an iterative algorithm that minimizes $R(w,b)$. The problem is that **the solution is not unique**. In fact, Rosenblatt posed the question of finding at least one solution. We will proceed with a gradient descent with respect to the free parameters $\Theta = (w,b)$.

5.2.2 Gradient Descent Algorithm

The gradient descent algorithm can be described as follows (see Figure 40, where k indicates the k-th step of the calculation):

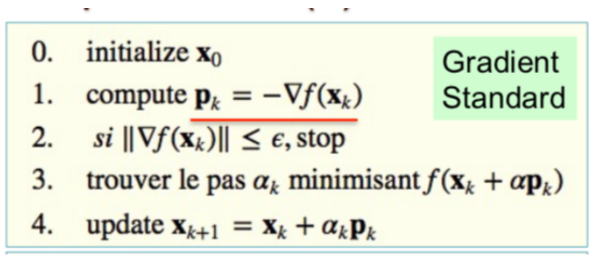
— We start with an initial value Θ^0.
— Next, we calculate the gradient of the function to be minimized:

$$(\nabla R(\Theta))_k = \frac{\partial R}{\partial \Theta_k} \tag{41}$$

The derivative with respect to a unit direction $\vec{v}$ is then the dot product:

$$\frac{\partial R}{\partial \vec{v}} = \vec{v}.\vec{\nabla} R(\Theta) \tag{42}$$

To maximize convergence speed, $\vec{v}$ should be collinear and opposite to the gradient.



```
0. initialize $\mathbf{x}_0$
1. compute $\mathbf{p}_k = -\nabla f(\mathbf{x}_k)$
2. si $\|\nabla f(\mathbf{x}_k)\| \leq \epsilon$, stop
3. trouver le pas $\alpha_k$ minimisant $f(\mathbf{x}_k + \alpha\mathbf{p}_k)$
4. update $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k\mathbf{p}_k$
```

Gradient Standard

Figure 40 – Diagram of a simple gradient descent.

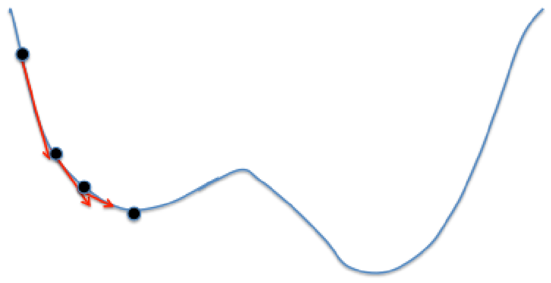


Figure 41 – Schematic of a gradient descent. We can get "stuck" in a local minimum.

In practice, the evolution of the values of Θ is represented as shown in Figure 41 in one dimension. As can be seen in this example, the algorithm can get stuck in a local minimum, but that's not the focus here; we will see improvements to this method in later courses [24].

So, let's calculate the gradients:

$$(\nabla R(\Theta))_k = \begin{cases} \sum_{i\in\mathcal{M}}(-y_i)\ x_{i,k} & \text{for } w_k \\ \sum_{i\in\mathcal{M}}(-y_i) & \text{for } b \end{cases} \tag{43}$$

(Note: The index k can be omitted for vector notation.)

However, we can either calculate the sum $\sum_{i\in\mathcal{M}}$ directly at each "step" of the calculation or **visit the points one by one**. This latter method had already been introduced by Rosenblatt. It is, in fact, the **stochastic gradient**, which had not been formalized in this way at the time. That means that each misclassified point already has a cost:

$$r_i(w,b) = (-y_i)\ (\langle w, x_i\rangle + b) \tag{44}$$

with a gradient:

$$\nabla r_i(\Theta) = \begin{cases} (-y_i)\ x_i \\ (-y_i) \end{cases} \tag{45}$$

So, **every time we have a new data point, we can update the parameters** Θ:

$$\boxed{\Theta^{(s+1)} = \Theta^{(s)} - \alpha^{(s)}\nabla r_i(\Theta)} \tag{46}$$

Thus, in the case we are considering here:

$$\begin{aligned} w^{(s+1)} &= w^{(s)} + \alpha^{(s)} y_i x_i \\ b^{(s+1)} &= b^{(s)} + \alpha^{(s)} y_i \end{aligned} \tag{47}$$

Both algorithms (standard and stochastic) are schematically compared in Figure 42 in

24. Note: In 2018, there was a seminar dedicated to new "Lazy" Gradient Descent methods.

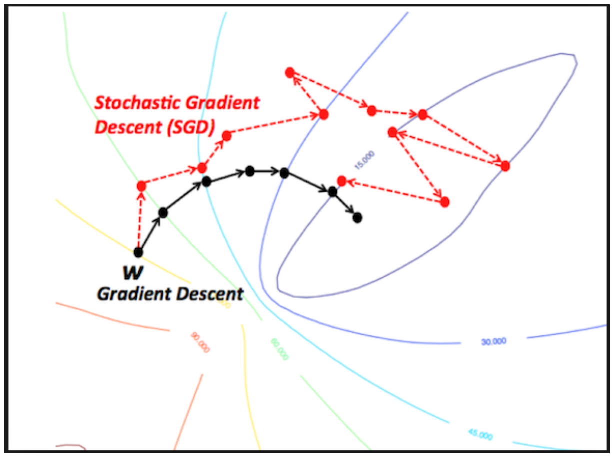


FIGURE 42 – Schematic comparison of global (Batch) gradient descent and stochastic gradient descent.

the case of searching for a minimum in 2D.

An observation known as the **Hebbian Rule** (named after Donald Hebb, a Canadian psychologist and neuropsychologist, 1904-1985) in biology and in computer science notes that **the weight w is increased if the input x and the output y** (which is also the input of a neuron in a subsequent layer) **are positively correlated**.

Now, the question that comes to mind is: how does it converge?

Theorem 1 *The Perceptron converges to a separating hyperplane if the data is separable.*

An example of **non-separability** is the **XOR** function. This was noticed very quickly and weakened the results of Rosenblatt's Perceptron[25].

Let's assume that we are in a separable case, meaning that there is a possible solution. Does the Perceptron converge to this solution? Yes... but.

Proof 1. So, there is a solution by hypothesis. Let $x^* = (x, 1)$ and $\Theta = (w, b)$, then:

$$\exists \Theta_* \; / \quad \forall i \quad y_i \langle \Theta_*, x_i^* \rangle > 0 \tag{48}$$

25. Note: In the 1970s, there was great disappointment in the community because all linear classifiers were affected (Fisher, Perceptron, etc.), and it seemed impossible to solve non-linear problems.

We can normalize $||x^*|| = 1$, and the condition remains the same. Now, once Θ_* is known, we can normalize it so that the condition changes to:

$$\exists \Theta_* \ / \quad \min_i \left(y_i \langle \Theta_*, x_i^* \rangle\right) = 1 \tag{49}$$

If, for the moment, α is a constant, then the rule for modifying Θ becomes:

$$\Theta^{(s+1)} = \Theta^{(s)} + \alpha y_i x_i^* \tag{50}$$

So:

$$\begin{align} ||\Theta^{(s+1)} - \Theta_*||^2 &= ||\Theta^{(s)} - \Theta_* + \alpha y_i x_i^*||^2 \tag{51} \\ &= ||\Theta^{(s)} - \Theta_*||^2 + \alpha^2 y_i^2 ||x_i^*||^2 + 2\alpha y_i \langle (\Theta^{(s)} - \Theta_*), x_i^* \rangle \tag{52} \end{align}$$

Note that the iteration is done over the samples (x_i, y_i) that are *misclassified*, meaning that at step s:

$$y_i \langle \Theta^{(s)}, x_i^* \rangle < 0 \tag{53}$$

So, by using the constraint on Θ_* and noting $y_i = \pm 1$ and taking $\alpha = 1$ (the gradient step or learning rate) to minimize the upper bound, then:

$$\begin{align} ||\Theta^{(s+1)} - \Theta_*||^2 &\leq ||\Theta^{(s)} - \Theta_*||^2 + \alpha^2 y_i^2 - 2\alpha y_i \langle \Theta_*, x_i^* \rangle \tag{54} \\ &\leq ||\Theta^{(s)} - \Theta_*||^2 + \alpha^2 y_i^2 - 2\alpha \tag{55} \\ &\leq ||\Theta^{(s)} - \Theta_*||^2 + \alpha^2 - 2\alpha \tag{56} \\ &\leq ||\Theta^{(s)} - \Theta_*||^2 - 1 \tag{57} \end{align}$$

So, at step $s + 1$, we gain on the squared distance to the limit by 1 unit. It takes N iterations for:

$$||\Theta^{(N)} - \Theta_*||^2 < 1 \tag{58}$$

with N given by:

$$N = \lceil ||\Theta^{(0)} - \Theta_*||^2 \rceil \tag{59}$$

and in this case, $\Theta^{(N)}$ defines a separating plane.

Indeed,

$$\forall i \qquad y_i\langle\Theta^{(N)}, x_i^*\rangle = y_i\langle\Theta^{(N)} - \Theta_*, x_i^*\rangle + y_i\langle\Theta_*, x_i^*\rangle \tag{60}$$

Now,

$$|y_i\langle\Theta^{(N)} - \Theta_*, x_i^*\rangle| = |\langle\Theta^{(N)} - \Theta_*, x_i^*\rangle| \leq ||x_i^*|| \; ||\Theta^{(N)} - \Theta_*|| \leq 1 \tag{61}$$

and

$$y_i\langle\Theta_*, x_i^*\rangle > 1 \tag{62}$$

so

$$\forall i \quad y_i\langle\Theta^{(N)}, x_i^*\rangle > 0 \tag{63}$$

which indeed defines a separating plane. ■

So, $\Theta^{(N)}$ is not necessarily equal to Θ^* because there is no uniqueness of the solution to the separating hyperplane, but it is one obtained in N steps. However, we do not have *a priori* control over the choice of the final separating plane, and as we will see, they are not all equivalent. Furthermore, in the non-separable case, it can be shown that the parameters will exhibit a potentially very long cyclic behavior, and the diagnosis that the algorithm is not converging may not be obvious.

5.2.3 Regularization

It has been mentioned several times that neural networks (implicitly referring to *deep* networks) have a considerable number of parameters, and given the relatively small number of samples, we should not be able to make them learn the weights. However, we find that we can converge the learning algorithm towards a solution that does not overfit or overfits very little, meaning it is robust enough for generalization. This is due to **regularization**, and this is precisely what is missing in the Perceptron algorithm that we just implemented in the previous section.

Let's introduce the important concept of **margin**, which will define the notion of **regularization**, and we will see the correspondence with the **penalty** on weights. You can refer to the 2018 course on *Support Vector Machine Classification* for more details.

The idea, proposed by Vladimir Vapnik in 1990 (born 1936), a Russian mathemati-

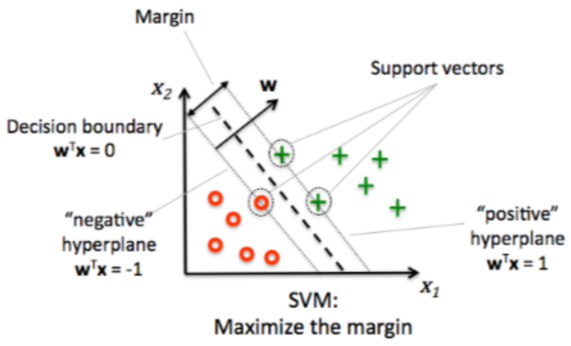


FIGURE 43 – Schematic representation of the margin concept to find the hyperplane least sensitive to sample variability near the boundary.

cian, is as follows: we need to **choose the hyperplane that is most robust to small changes in the training data** (see Figure 43). In other words, we choose the hyperplane that is "farthest" from both classes: x_i^+ and x_i^-.

Note that on the hyperplane $\langle w, x\rangle + b = 0$, and outside of it $\langle w, x\rangle + b = c \neq 0$. And on either side of the plane, we can find the points x_1 and x_2 closest to the hyperplane of both classes and such that the hyperplane is in the middle (meaning that the point $(x_1 + x_2)/2$ is on the hyperplane). Therefore, if we denote $m||w||$ as the distance of x_1 and x_2 to the plane, as they are on either side, in general we have:

$$\begin{cases} \langle w, x_1\rangle + b & = m||w|| \\ \langle w, x_2\rangle + b & = -m||w|| \end{cases} \tag{64}$$

Now, we can rescale the x values such that the distance between x_1 and x_2 is equal to 2. Thus:

$$\boxed{||w|| = \frac{1}{m}} \tag{65}$$

So, **maximizing the "2m" margin (and therefore m) is equivalent to minimizing** $||w||$. This leads to a truly important result: **when we minimize the norm of the network's weights, we tend to get a network that is more robust for generalization**.

5.2.4 SVM: Support Vector Machine

Given that w is a linear combination of x_i, the Perceptron problem can be reformulated as follows: find (w, b) such that:

$$\boxed{y_i \times (\langle w, x_i \rangle + b) \geq 1; \text{ and } \quad \min(||w||^2)} \tag{66}$$

During the 2018 course, we solved this constrained problem by introducing a Lagrangian with Lagrange multipliers α_i:

$$\boxed{L(w, b, \alpha) = \frac{1}{2}||w||^2 + \sum_i \alpha_i (1 - y_i \times (\langle w, x_i \rangle + b))} \tag{67}$$

We need to solve the case where the problem is not strictly separable due to *outliers* (a pathological case of poor classification). This is solved by introducing a Perceptron-like penalty for cases of misclassification. Vapnik suggests defining the error on the margin as how much the misclassified points/samples need to be moved to place them beyond the margin for correct reclassification. So, if we denote $\xi_i \geq 0$ as the distance for sample i that needs to be moved, then:

$$y_i \times (\langle w, x_i \rangle + b) \geq 1 - \xi_i \quad \text{(Soft SVM)} \tag{68}$$

and the constraint becomes:

$$\min(||w||^2 + C \sum_i \xi_i) \tag{69}$$

So, SVM resembles the Perceptron, but the constraint on $||w||^2$ changes all the properties of the classifier: it is **more robust**, and since the constraint is convex, it converges to a **unique solution**.

Note that we could have chosen a constraint $\sum_i \xi_i^2$, but then the very badly classified points would have a very strong influence on the found solution; by taking $\sum_i \xi_i$, we penalize the cases of very poor classification without having too much influence.

5.2.5 Assessment of the Perceptron

Through this simple example of the Perceptron and its extension to the Support Vector Machine, important concepts have emerged:

— gradient descent (GD)

— its variant, stochastic gradient descent (SGD) (although in this specific case, randomness did not appear as an essential ingredient, thanks to the convexity of the simple problem. We will revisit this in a later course.)

— convergence issues

— regularization problems.

However, the **approximation problem** remains. We manage to make the algorithm converge, but **what is the size of the set of functions that can be approximated using the Perceptron?** We know that XOR cannot fit into this class.

6. Multi-layer Architecture: Part I

6.1 Introduction

In the introduction, S. Mallat refers to the depression in the community during the 1970s[26]. During those years, linear classifiers were disqualified, and then it was realized that the path of multi-layers had to solve the problem of non-linear classifications. All of this was well formalized in the 1980s and 1990s.

We will study the **network with 1 hidden layer** or, as some call it, "the 2-layer network" (see Figure 44). We will show that **this network can approximate any function** (linear or not). This is the **Universal Approximation Theorem** that we will demonstrate from two different perspectives. Through the proof of this theorem, we will see that:

— While the proof of this theorem can be considered a beautiful piece of mathematics, it still does not solve the problem as outlined last year. The **curse of dimensionality** is still not far off.

26. NDJE: This is the "first ice age". There was a second one in the 1990s due to gradients becoming zero or infinite as the number of layers increased.

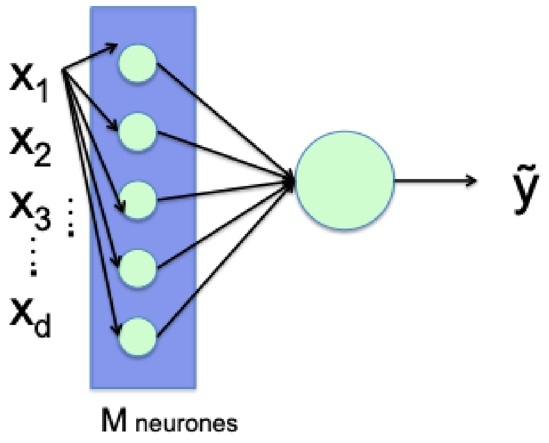


FIGURE 44 – Schematic of a neural network with 1 hidden layer or 2 layers (including the input layer and the hidden layer). *Note that the number of hidden neurons is also denoted as K.*

— Nevertheless, this theorem is interesting because it helps understand the nature of the problem. In particular, when we perform **approximation, we use the regularity of the function**, and thus, we see which mathematical tool we use in this case: **the theory of approximation and Fourier analysis are absolutely central here** to understand the approximation capacity of this network.

6.2 Expression of the Network's Output

In the architecture shown in Figure 44 (note that the notation has changed slightly from 2018, but nothing fundamental),

$$\tilde{y} = \sigma\left[\sum_{k=1}^{K} c_k \ \sigma\left[\sum_{m=1}^{d} w_{k,m} x_m + b_k\right] + b'_k\right] = \tilde{f}(x) \tag{70}$$

where $\sigma(x)$ is a non-linearity (ReLU, sigmoid, tanh, sign, etc.). What we now observe is that in neural networks, if we change the non-linearity function (and many experiments have been done), the architecture, if properly configured, will still work when retrained. It just needs to have a non-polynomial non-linearity. The **ReLU function is particularly**

suitable because its derivative is almost everywhere defined and bounded, making gradient descent work well. By doing this, we have **abandoned any over-interpretation of the types of non-linearity**, as has often been done in the past.

Note that each neuron k in the hidden layer responds as follows:

$$x_k \equiv \sigma \left[\sum_{m=1}^{d} w_{k,m} x_m + b_k \right] \tag{71}$$

and this is nothing but the application of a **data separation by the hyperplane** (w_k, b_k). Thus, **in the hidden layer, we obtain the collection of K data separation hyperplanes** in one pass, which lets you imagine the power of selection. We can write the previous relationship for hidden neuron k in a matrix form:

$$x_k = \sigma \left[W_k x + b_k \right] \tag{72}$$

and this generalizes in a multi-layer network where the output of layer $j+1$ depends on the output of the previous layer j:

$$\vec{x}_{j+1} = \sigma \left[W_j \vec{x}_j + \vec{b}_j \right] \tag{73}$$

or, using another notation:

$$\boxed{\mathbf{x}_{j+1} = \sigma \left[\mathbf{W}_j \mathbf{x}_j + \mathbf{b}_j \right]} \tag{74}$$

In the subsequent courses, we generally omit the notation with arrows or the use of bold letters for vectors (although we will use it at times), but one should still visualize this vector/matrix concept (note that the non-linearity is applied component-wise within the brackets).

Denoting Θ as the set of network parameters, which includes all $(W_k, b_k, c_k, b'_k)_{k \leq K}$, we are interested in the **class of all functions $\tilde{f}_\Theta$**:

$$\mathcal{H} = \left\{ \tilde{f}_\Theta \ / \ \forall \ \Theta \right\} \tag{75}$$

and we wonder if this **class is large enough to approximate the function $f(x)$ of interest?** If not, we will have a bias error.

We have n **labeled samples** $\{x_i, y_i = f(x_i)\}_{i \leq n}$, and as we saw previously, we have a problem that can be divided into three parts (**approximation**, **estimation**, and **optimization**):

— First, we will try to search within $\mathcal{H}$ for a specific function $\tilde{f}_{\Theta^*}$ such that $\tilde{f}_{\Theta^*} \approx f(x)$. This is a question of **approximation**.

— Second, even if $\tilde{f}_{\Theta^*}$ exists, **can it be estimated from n examples?**

— Finally, even if it were possible to estimate it, we want to obtain a "good" approximation, so we have an **optimization problem**.

In the following, we will tackle the "approximation" part, knowing that "estimation" is straightforward in this case (due to convexity), and the "optimization" part remains to be addressed. In fact, the most challenging problem is indeed approximation.

6.2.1 The Case of Boolean Functions

A priori, we have a function of the type:

$$x \in \mathbb{R}^d \xrightarrow{f} y \in \mathbb{R} \tag{76}$$

But here, we will approximate each component of x with a q-bit (*note that the notation for a bit here is* ± 1), the same goes for y. So, initially, let's consider functions of the type:

$$x \in (\pm 1)^{q \times d} \xrightarrow{f} y \in (\pm 1)^q \tag{77}$$

If q is large enough, we can know real numbers with good numerical precision. Note that we can decompose the function f into q functions that give 1 bit of y. So, we simplify the problem by studying functions of the type:

$$x \in (\pm 1)^{q \times d} \xrightarrow{f} y \in (\pm 1) \tag{78}$$

We can view this problem as a 2-class classification with input vectors of $q \times d = d'$ quantized dimensions.

Theorem 2 *Let f be a function such that*

$$x \in (\pm 1)^{d'} \xrightarrow{f} y \in (\pm 1) \tag{79}$$

then, for any d', there exists a 2-layer network (or 1 hidden layer) such that $f \in \mathcal{H}$.

Proof 2. We will construct the right neural network and see how many hidden neurons we need. Let's try to build a network by focusing on the set of x defined by

$$\mathcal{A} = \{x \;/\; f(x) = 1\} \tag{80}$$

Note that x has $2^{d'}$ possible values. Take K elements from this set, denoted as

$$\{w_k\}_{k \leq K} \quad \text{with} \quad w_k = (\pm 1)^{d'} \quad \text{and} \quad f(w_k) = 1 \tag{81}$$

And for the nonlinearities, we will take the "sign" function. Now consider the relationship between the input x and the first layer:

$$\sigma(w_k.x + b_k) \tag{82}$$

By analyzing the bits of x and w_k, the dot product is equal to

$$w_k.x = \left| \begin{array}{rl} d' & \text{if} \quad x = w_k \\ \leq d' - 2 & \text{if} \quad x \neq w_k \end{array} \right. \tag{83}$$

And if we take $b_k = 1 - d'$, then

$$\sigma(w_k.x + b_k) = \left| \begin{array}{rl} 1 & \text{if} \quad x = w_k \\ -1 & \text{if} \quad x \neq w_k \end{array} \right. \tag{84}$$

So, each neuron k detects whether the input x is equal to w_k or not. We can express $f(x)$ as a sequence of OR operations:

$$f(x) = 1 \text{ if } (x = w_1) \| (x = w_2) \| \ldots \| (x = w_K) \tag{85}$$

So, we need to perform a logical disjunction (OR) of these K equalities. In fact, we can do it like this:

$$\boxed{f(x) = \sigma\left(\sum_{k=1}^{K} \sigma(w_k.x + 1 - d') + K - 1\right) = \left|\begin{array}{rl} 1 & \text{if} \quad \exists k \ / \ x = w_k \\ -1 & \text{if} \quad \forall k,\ x \neq w_k \end{array}\right.} \tag{86}$$

■

In fact, we have constructed a representation of a Boolean function. However, to represent all x such that $f(x) = 1$ as q-bits in d-dimensional space, **we will typically need** $2^{d'-1} = K$ **neurons**, assuming that half of the vectors belong to class 1. Otherwise, we can certainly say that $K \approx O(2^{d'})$. So, we need an **exponential number of neurons**.

In the end, this type of network is **a large memory** that stores vectors (w_k) for which the function is equal to 1. For classification, it simply compares the input to these encoded vectors. **These networks, therefore, seem rather coarse**, even devoid of any form of complexity. However, it is known that these single-hidden-layer neurons are satisfactory for certain problems but not too complex ones, meaning they cannot be used for image classification. For that, we need to go beyond this simple description of Boolean functions.

6.2.2 Using Function Regularity

In the previous construction, we were content with storing vectors such that $f(x) = 1$, **without considering the regularity of the function**. So, first, we must genuinely consider $x \in \mathbb{R}^d$ as a continuous variable (i.e., not quantized), and second, analyze the regularity of $f(x)$. It's in this context of continuity that the **Universal Approximation Theorem** emerges. We will simplify the expression:

$$\tilde{f}(x) = \sigma\left[\sum_{k=1}^{K} c_k \ \sigma\left[\sum_{m=1}^{d} w_{k,m} x_m + b_k\right] + b'_k\right] \tag{87}$$

by removing the last unnecessary non-linearity here. Then we attempt to approximate $\tilde{f}(x)$ as follows:

$$\tilde{f}(x) = \sum_{k=1}^{K} c_k \; \sigma \left[\sum_{m=1}^{d} w_{k,m} x_m + b_k \right] = \sum_{k=1}^{K} c_k \; \sigma \left[w_k.x + b_k \right] \tag{88}$$

Note that the terms $\sigma\left[w_k.x + b_k\right]$ are functions referred to as **"ridge functions"**. In general, a "ridge" function is one that satisfies:

$$f : \mathbb{R}^d \longrightarrow \mathbb{R} \text{ such that } \exists g_a : \mathbb{R} \longrightarrow \mathbb{R} \text{ such that } f(x) = g(a.x) \tag{89}$$

Then,

$$\forall x \in \mathbb{R}^d \;/\; a.x = c \in \mathbb{R} \Rightarrow f(x) = g(c) \tag{90}$$

So, a ridge function is constant on the hyperplane $a.x = c$**.** Thus, $\sigma\left[w_k.x + b_k\right]$ is constant for x belonging to hyperplanes parallel to the hyperplane (w_k, b_k), and it only varies if x has a component collinear with w_k.

Therefore, the problem is to decompose $f(x)$ (find an approximation) into K functions (ridges) that vary only in one direction each. The ultimate question will be how many directions (i.e., values of K) are needed to approximate $f(x)$?

Let the set of functions be[27]:

$$\mathcal{M}_\sigma = \text{Vect} \left\{ \sigma\left[w.x + b\right] / w \in \mathbb{R}^d, \; b \in \mathbb{R} \right\} \tag{91}$$

Is it large enough to approximate the function f? In other words, **is** $\mathcal{M}_\sigma$ **dense in the set of continuous functions** $C(\mathbb{R}^d)$? This is the minimum level of regularity we impose.

So, we want to approximate $f(x)$ with a linear combination of elements from $\mathcal{M}_\sigma$ as precisely as possible. Therefore, we need a criterion, a distance. For this purpose, we will consider compact sets Ω in $\mathbb{R}^d$ (essentially bounded sets), and we want:

$$\forall \epsilon > 0, \; \exists \tilde{f} \in \mathcal{M}_\sigma \;/\; \forall \Omega \subset \mathbb{R}^d, \; \forall x \in \Omega \quad |f(x) - \tilde{f}(x)| \leq \epsilon \tag{92}$$

This is the **metric of uniform convergence on a compact set**, which is a **particular**

27. nb. "linear combination" = Vect

topology. These aspects were extensively studied between 1987 and 1993:

— Hecht & Nielsen, who used a result by Kolmogorov that, in fact, should not have applied, showed a new way to pose the problem and brought a certain renewal to the problem.

— Gallant & White then asked whether it is possible to construct a particular sigmoid function that could satisfy the approximation result.

— Cybenko in '89 showed that the result holds for any sigmoid.

— Liensko, Lin, Pinkus, and Schocken ('93) showed that the result generalizes to any nonlinear function σ as long as it's not a polynomial.

Finally, S. Mallat mentions the review paper from 1999 that presented the state of the art at that time: **Allan Pinkus** *"Approximation theory of MLP model in Neural Network"*[28].

6.3 Universality Theorem of a 1-hidden layer network

Theorem 3 *Let $\sigma \in C(\mathbb{R})$, then $\mathcal{M}_\sigma$ is dense for uniform convergence on a compact set if and only if σ is not a polynomial.*

We will proceed with a **constructive** proof because through this construction, we aim to understand: how it works, why it is challenging, how it relates to **the regularity of the function** to approximate, especially **the smoother the function, the fewer hidden neurons are needed**, and how **the curse of dimensionality manifests itself except in a few special cases**, etc.

If σ is a polynomial of degree m, let's see what happens in $d = 1$. In this case:

$$\tilde{f}(x) = \sum_{k=1}^{K} c_k \ \sigma(w_k x + b_k) \tag{93}$$

is also a polynomial of degree m. However, this is not sufficient to approximate a continuous function on an interval. We can verify this with a polynomial of degree $m+1$; if we want to approximate it with a polynomial of degree m, there will be a significant error.

28. http://www2.math.technion.ac.il/~{}pinkus/papers/acta.pdf

For example, on the bounded interval $\Omega = [0, 1]$, consider the continuous function:

$$f(x) = x^5 \tag{94}$$

and attempt to fit it with a degree 4 polynomial. The residuals are shown in Figure 45. We cannot achieve uniform convergence with arbitrarily small error on $[0, 1]$. Therefore, we observe that **this approach does not work because we remain within the class of polynomials if σ is a polynomial**.

In fact, let's specify the problem that arises: in the case of using an MLP, we do not actually know f; we only have labeled samples, and we want to fix σ *a priori* (in the above case, we fix m as the polynomial degree) to satisfy criteria for all f belonging to a certain class of functions, here continuous functions on $\mathbb{R}^d$, i.e., $C(\mathbb{R}^d)$. Therefore, let's consider **a non-polynomial** σ. S. Mallat mentions that there are many proof strategies. The one he proposes involves *sine/cosine* functions because the mathematically well-understood **ridge functions** that vary in only one dimension are the basis for **Fourier** functions. We will select sinusoidal functions in $\mathbb{R}^d$ that vary in only one dimension (see Figure 46).

6.3.1 Fourier Basis

Lemma 1 *Any function belonging to $C(\mathbb{R}^d)$ can be approximated with arbitrary precision using a Fourier basis, that is,*

$$\begin{aligned}&\forall f \in C(\mathbb{R}^d), \forall \Omega(compact) \subset \mathbb{R}^d, \forall \epsilon > 0, \exists K \ and\ \{w_k\}_{k \leq K} \\ &\qquad s.t.:\forall x \in \Omega \quad |f(x) - \sum_{k=1}^{K} (\alpha_k \cos(w_k \cdot x) + \beta_k \sin(w_k \cdot x))| \leq \epsilon\end{aligned} \tag{95}$$

Through this lemma, we establish a connection with **Harmonic Analysis**, which will help us understand how the number of ridge functions (cf. K) is directly related to the regularity of function f.

Imagine for a moment that this lemma is proven. If we can subsequently write a

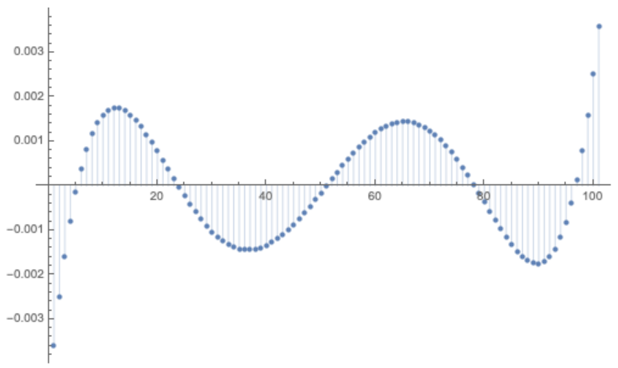


FIGURE 45 – Residuals of fitting the function x^5 with a degree 4 polynomial.

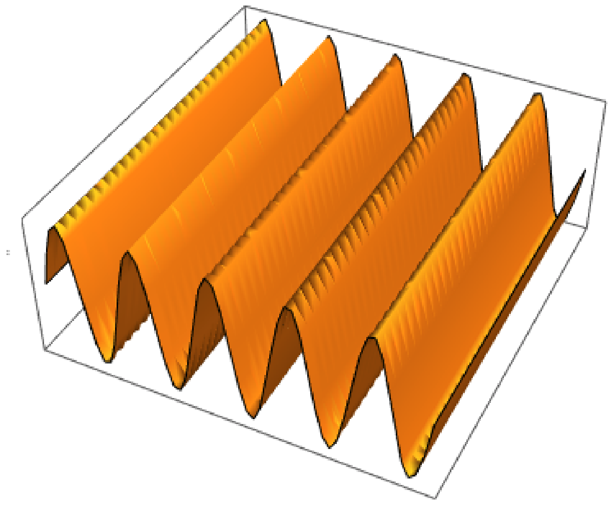


FIGURE 46 – Ridge sinusoidal function varying in only one dimension.

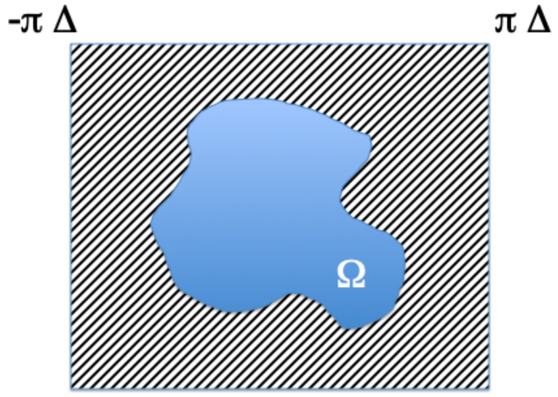


FIGURE 47 – Embedding of the compact Ω into a cubic box of size $[-\pi\Delta, \pi\Delta]^d$.

decomposition of *sine/cosine* functions in terms of ridge functions σ as follows:

$$\cos(t) = \sum_j \lambda_j^c \; \sigma(\gamma_j^c t + \delta_j^c) \tag{96}$$

$$\sin(t) = \sum_j \lambda_j^s \; \sigma(\gamma_j^s t + \delta_j^s) \tag{97}$$

then we will have made progress.

Let's move on to the proof of the lemma by relating K to the regularity of f. So, we work on a compact set $\Omega \subset \mathbb{R}^d$. Since it is **bounded**, we can embed it in a larger box such that (see Figure 47):

$$\Omega \subset [-\pi\Delta, +\pi\Delta]^d \tag{98}$$

Next, we define a finite-energy Fourier basis on the large box $L^2([-\pi\Delta, +\pi\Delta]^d)$.

A quick reminder in $d = 1$:

$$L^2([-\pi\Delta, +\pi\Delta]) = \left\{ f \;/\; ||f||^2 = \int_{-\pi\Delta}^{\pi\Delta} |f(x)|^2 dx < \infty \right\} \tag{99}$$

On $L^2([-\pi\Delta, +\pi\Delta])$, we know that the family

$$\left\{ e^{inx/\Delta} \right\}_{n\in\mathbb{Z}} \tag{100}$$

is an orthogonal basis, with the inner product defined as

$$\langle f, g\rangle = \int_{-\pi\Delta}^{\pi\Delta} f(x)g^*(x)dx \tag{101}$$

Now, in d dimensions, $x \in \mathbb{R}^d = (x_1, \ldots, x_d)$, and the set of vectors

$$\left\{e^{in_1x_1/\Delta}.e^{in_2x_2/\Delta}.(\ldots).e^{in_dx_d/\Delta}\right\}_{\{n_1,\ldots,n_d\}\in\mathbb{Z}^d} = \left\{e^{iw.x/\Delta}\right\}_{w\in\mathbb{Z}^d} \tag{102}$$

form a **orthogonal basis for** $L^2([-\pi\Delta, +\pi\Delta]^d)$ [29].

We want to approximate f on Ω (see Figure 47) which we have embedded in $[-\pi\Delta, +\pi\Delta]^d$. We will extend f to $\mathbb{R}^d$ by imposing a null condition outside of $[-\pi\Delta, +\pi\Delta]^d$ and, for example, by using linear interpolation on the complement of Ω in $[-\pi\Delta, +\pi\Delta]^d$. Thus, based on the preceding discussion, $f \in L^2([-\pi\Delta, +\pi\Delta]^d)$ can be decomposed in the Fourier basis as follows [30]:

$$f(x) = \frac{1}{(2\pi\Delta)^d}\sum_{w\in\mathbb{Z}^d}\hat{f}(w)e^{iw.x/\Delta} \tag{103}$$

However, this requires **an infinite number of frequencies**, so we would like to know if we can take a **limited number** (this is a form of sparsity) and still obtain a good approximation of f.

The $w \in \mathbb{Z}^d$ are actually **the weights of the hidden neurons**; in 2D, they tile space as shown in Figure 48 (integer intersections). By **constraining** them to the smallest weights (red circle), we recover the **Perceptron regularization**:

$$||w|| \leq C \tag{104}$$

This restriction is very natural because we want to approximate **smooth functions** where **low frequencies** dominate. We will then attempt to approximate function f using the truncated Fourier basis and examine the error norm.

29. Note: The proof proceeds by showing that the 1D version is orthogonal, the density is achieved by the Poisson kernel, and the d-dimensional version is formed by the product of the 1D bases.
30. The normalization coefficient is not important.

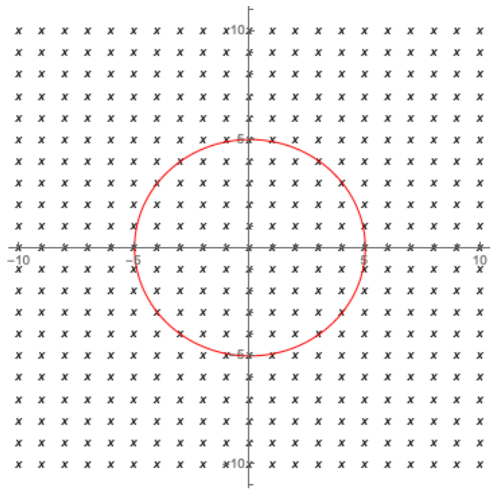


FIGURE 48 – Illustration of the weights $w = \{n_1, \ldots, n_d\} \in \mathbb{Z}^d$, and focusing on weights $||w|| \leq C$ is equivalent to using Perceptron-type regularization.

6.3.2 Approximation of Sinusoids by σ

So, the next step is to approximate sinusoids using the non-linearity. **We will do this with the ReLU**. Therefore, we need to consider cases of:

$$\begin{cases} \cos(w.x) \\ \sin(w.x) \end{cases} \quad \text{with } |w| < C \text{ and } x \in [-\pi\Delta, +\pi\Delta]^d \tag{105}$$

Note that the dot product $w.x$ (the argument of cosine or sine) is bounded by:

$$|w.x| < |w|.||x||_{L2} < |w|.||x||_{L1} < C\pi\Delta d \tag{106}$$

Let's consider cosine (here, we are dealing with scalars, so in 1D). We want:

$$|\cos(t) - \sum_j \lambda_j^c \, \sigma(\gamma_j^c t + \delta_j^c)| \leq \epsilon/K \tag{107}$$

(The bound comes from the fact that the approximation of f by the sum of K cosines and sines is bounded by ϵ; we ignore the factor of 2). How many non-linearities do we need?

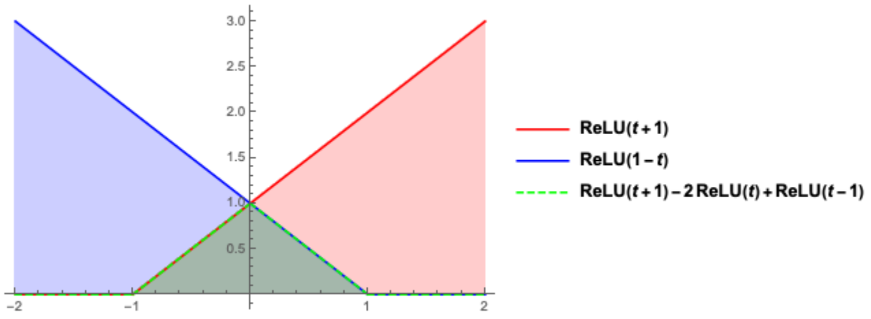


FIGURE 49 – With a linear combination of ReLU, we can obtain a "triangle" function (in green).

Observe that with **ReLU, we can create a Triangle function** through a linear combination, as shown in Figure 49. We know (from the 2018 lecture) that **the Nyquist theorem** tells us that this triangle generates the vector space of linear splines. To convince yourself, in Figure 50, you can easily see that the sum of the red and green triangles gives the blue curve. Therefore, triangles can precisely approximate a piecewise linear function, which can approximate the cosine to a certain level of fixed precision.

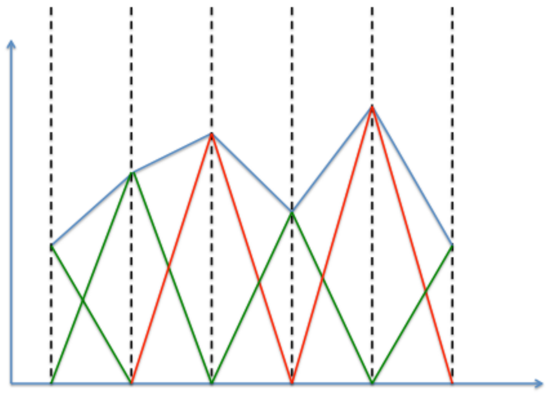


FIGURE 50 – Approximation of a piecewise affine function using properly normalized "triangle" functions.

So, by using ReLU and their translations, we can achieve any piecewise linear

approximation of the cosine function.

Note that **biases are used in this operation to translate the ReLU functions**. However, how many ReLU functions do we need? In fact, we need as many as the number of biases, which is the same as the number of samples needed to obtain an approximation of $\epsilon' = \epsilon/K$ accuracy [31].

We need $2K$ sines and cosines, and each sine/cosine requires $2\pi\Delta dC/\epsilon'$ samples (domain size divided by error), so the number of ReLU functions is (order of magnitude):

$$N_\sigma \approx K^2\Delta dC/\epsilon \approx C^{2d+1}\frac{d\Delta}{\epsilon} \tag{108}$$

In this formula, ultimately Δ is the domain size, and d is the dimension; these two factors are not critical. On the other hand, we have the **crucial factor C^{2d+1} with $C(\epsilon)$ governed by the regularity of the function which determines the high-frequency cut-off.**

In the next section, we will analyze how $C(\epsilon)$ varies with the regularity of the function. However, we won't be able to constrain it to decrease rapidly enough to counteract the exponential increase to avoid **the curse of dimensionality**. The only case that might work is when we can use **much fewer** frequencies, meaning we can be **sparse**. There are cases where this happens, such as when the problem is separable in different dimensions. We will also see the state of knowledge in the year 2000 on what can be learned as the number of layers increases: we will see that there was not much hope - "increasing the number of layers won't help!"

7. Multi-Layer Architecture: Part II

7.1 Introduction

Let's revisit the **Universal Approximation Theorem**, not because the demonstration in the previous section provides a construction of a Neural Network with 1 hidden layer, but rather because it highlights the limitations of this type of network and provides an

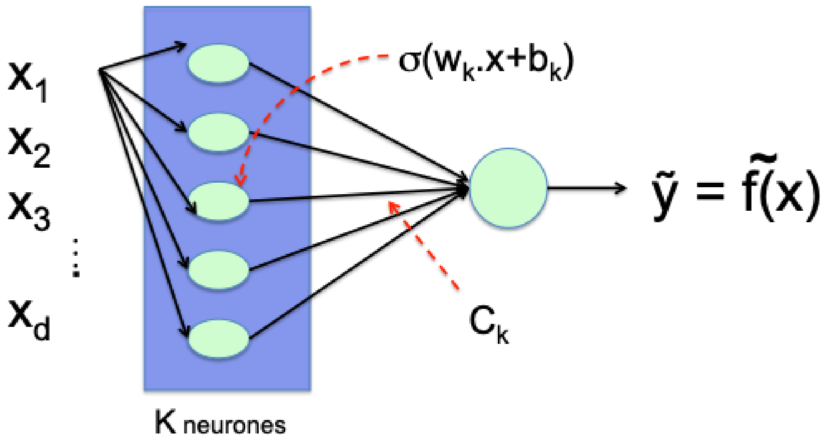


Figure 51 – Slightly modified diagram of Figure 44 with notations from the Universal Approximation Theorem.

understanding of the problem and the questions raised.

So, we start with the diagram in Figure 51 (note that the second non-linearity will not be necessary), and we ask: **can we approximate any continuous function on** $\mathbb{R}^d$ **depending on** K **(the number of hidden neurons)**?

The first-level answer is indeed **yes**, we can approximate any function, but in most cases, K will become enormous as soon as we require a good approximation. The second-level answer is K **will have to be enormous**, but this will depend on the **regularity of the function** $f(x)$.

It is the second level that will interest us because **it connects** K**, the number of neurons, and the regularity of the function we want to approximate, which we do not know *a priori***.

31. As a side note, ReLU is not crucial; we need to define from the non-linearity a function with compact support that plays the role of a triangle

7.2 Recall of the Universal Approximation Theorem

Let's give another formulation of the **Universal Approximation Theorem** (Section 6.3):

Theorem 4 *If $f \in C(\mathbb{R}^d)$, and let $\sigma \in C(\mathbb{R})$ be a non-polynomial function, then*

$$\forall \epsilon > 0,\ \exists \tilde{f} \in \mathcal{M}_\sigma \ / \ \forall \Omega \subset \mathbb{R}^d,\ \forall x \in \Omega \quad |f(x) - \tilde{f}(x)| \leq \epsilon \tag{109}$$

with

$$\boxed{\tilde{f}(x) = \sum_{k=1}^{K(\epsilon)} C_k\ \sigma(w_k.x + b_k)} \tag{110}$$

A reminder of the demonstration principle, which happens in two steps:

1. We decompose $f(x)$ according to a family of ridge functions of the sinusoidal type. Decomposition into a Fourier series where we identify the $\{w_k\}$ as frequency indices (considering that the compact Ω is bounded and can be embedded in a large box of volume $(2\Delta)^d$, which can be quantized), and the $\{b_k\}$ as translation indices of the sampling function σ[32]. We limit it to low frequencies $|w| < C$ to obtain an approximation, which gives the value of K.
2. Then we show that sinusoids can be decomposed into σ functions.

In conclusion, the number of σ functions required N_σ for the error on $f(x)$ to be ϵ is obtained as follows:

$$\boxed{|w| < C_\epsilon \Rightarrow K = C_\epsilon^d \Rightarrow N_\sigma \approx K^2 \Delta d C_\epsilon/\epsilon \approx C_\epsilon^{2d+1}\frac{d\Delta}{\epsilon}} \tag{111}$$

In fact, it is **the constraint on K that matters**. The question is, what is the value of the constant C_ϵ?

32. Note that we used a ReLU, but the complete proof shows that this result is independent of σ

7.3 Convergence of the Approximation $\tilde{f}$

We know that we have truncated the high frequencies to obtain the approximation $\tilde{f}(x)$, so

$$\tilde{f}(x) = \sum_{|w|<C_\epsilon} \hat{f}(w)\ e^{i\ 2\pi \frac{w.x}{\Delta}} \tag{112}$$

Furthermore, as we are working with an orthogonal basis, we can quickly deduce an error in the approximation:

$$f(x) - \tilde{f}(x) = \sum_{|w|\geq C_\epsilon} \hat{f}(w)\ e^{i\ 2\pi \frac{w.x}{\Delta}} \tag{113}$$

In L^2 norm, this translates to

$$\|f(x) - \tilde{f}(x)\|^2 = \frac{1}{(2\pi\Delta)^d} \int_{[-\pi\Delta,\pi\Delta]^d} |f(x) - \tilde{f}(x)|^2 dx \tag{114}$$

The error corresponds to the energy of all the coefficients omitted, specifically

$$\|f(x) - \tilde{f}(x)\|^2 = \frac{1}{(2\pi\Delta)^d} \sum_{|w|\geq C_\epsilon} |\hat{f}(w)|^2 \tag{115}$$

The regularity of the function f is reflected in the rate of decay of the Fourier coefficients [33]. This concept forms the basis of Harmonic and Functional Analysis.

Note that since $f \in L^2([-\pi\Delta, \pi\Delta]^d)$, the Fourier series converges. So, when $C_\epsilon \to \infty$, we know that the error tends to 0. There are no issues in that regard.

The real challenge, once again, is how to choose the value of K. We can relate **the *uniform* regularity of f to the decay of** $|\hat{f}(w)|$, which ultimately defines K.

7.4 Definition(s) of Regularity

7.4.1 Regularity in the Sense of Derivatives (Sobolev/Hilbert): Maiorov's Optimality Theorem

Let's consider the 1D case for illustration. For instance, we can define regularity simply by the first derivative, i.e., consider the function to be **differentiable** and that it

33. NDJE: Review the 2018 course on Lipschitz functions.

has **finite energy**, which means that

$$\int |f'(x)|^2 dx < \infty \tag{116}$$

Now, $\hat{f}'(w) = iw\hat{f}(w)$, and if we are on a compact set, the second constraint yields

$$\sum_w |w|^2 |\hat{f}(w)|^2 < \infty \tag{117}$$

So, it's not only the energy of the function f that needs to be constrained; furthermore, as the sum converges, we have

$$|w|^2 |\hat{f}(w)|^2 = o(1) \Rightarrow |\hat{f}(w)| = o(|w|^{-1}) \tag{118}$$

indicating the type of decay at infinity of the Fourier coefficients[34]. We can extend this notion by requiring constraints on derivatives up to order m, then we define **Sobolev regularity**:

$$\sum_w |w|^{2m} |\hat{f}(w)|^2 < \infty \Rightarrow |\hat{f}(w)| = o(|w|^{-m}) \tag{119}$$

Since the function f is square-integrable, we can express the constraint differently[35].

$$\sum_w (1 + |w|^{2m}) |\hat{f}(w)|^2 = A < \infty \tag{120}$$

That being said, the conclusion is that high-frequency energy will decay faster as m increases. So, let's revisit the error calculation:

$$\begin{aligned} \|f(x) - \tilde{f}(x)\|^2 &= \frac{1}{(2\pi\Delta)^d} \sum_{|w| \geq C_\epsilon} |\hat{f}(w)|^2 < \sum_{|w| \geq C_\epsilon} \frac{(1 + |w|^{2m})}{C_\epsilon^{2m}} |\hat{f}(w)|^2 \\ &< A / C_\epsilon^{2m} = \epsilon^2 \end{aligned} \tag{121}$$

34. NDJE: reminder $o(1) \ll 1$ for a mathematician.

35. NDJE: one also finds for L^2 functions the constraint $\|f\|^2_{L^2,m} = \sum_w (1 + |w|^2)^m |\hat{f}(w)|^2 = A < \infty$, which indicates that f is an element of the Hilbert space H^m (cf. L^2 functions satisfying this Fourier constraint), but the one given by S. Mallat works just as well.

The constraint on C_ϵ and K (the number of neurons) can then be written as

$$\boxed{C_\epsilon = A^{1/2m}\epsilon^{-1/m} \Rightarrow K = A^{d/2m}\epsilon^{-d/m}} \tag{122}$$

The number of coefficients to adjust grows as

$$N_\sigma \propto K^2 \propto \epsilon^{-2d/m} \tag{123}$$

Remember that we want to make ϵ as small as possible, so for K not to explode in high dimensions (cf. large d), we can only approximate functions of very high regularity (cf. large m). In other words, requiring regularity on derivatives of order m gives an apparent dimension of the problem determined by d/m.

In the years 1988-94, efforts were made to obtain better upper bounds for equation 121. Here, S. Mallat used sine/cosine functions instead of directly analyzing ReLU functions to sample the function f, so it's not optimal but intentional on his part for clarity of presentation. The answer from these studies is that YES, instead of requiring K^2 coefficients, we can very well get by with just K coefficients. But this doesn't change the conceptual picture because $d \approx 10^6$. With that said, let's provide the definitive form of the theorem on Sobolev functions.

Theorem 5 ***Maiorov*** *(1999): If f has m Sobolev derivatives, then there exist sigmoid functions such that the total number of control coefficients is of the order of*

$$\boxed{K \approx \epsilon^{-(d-1)/m}} \tag{124}$$

But what's very nice is that **WE CAN'T DO BETTER!** That is to say, there is no (linear or non-linear) scheme that can beat this scaling law. So, we can say **"end of the story"**, we've found the universal approximation machine. **But the problem** remains the same: if we want to **reduce the error by 2**, then we must multiply the number of neurons by a colossal factor according to the law:

$$\epsilon \to \epsilon/2 \Rightarrow K \to 2^{(d-1)/m}\ K \tag{125}$$

So, this result, although mathematically elegant, is entirely impractical except in cases of reduced dimensionality! We need to consider other types of regularity much greater than and beyond Fourier Analysis for cases of very high dimensionality that would make the result less pessimistic.

7.4.2 Other Types of Regularity

A comment from S. Mallat: there is a "tricky" aspect of mathematics in which we arrive at theorems like Maiorov's, and we tend to forget the assumptions (not of the theorem itself) of the general conceptual framework in which they are embedded. Thus, the consequences of Maiorov's theorem have led the vast majority of people working in the field to consider that single or multi-layer neural networks will not be a future subject.

Therefore, it is better to revisit the basic questions that gave rise to the hypothesis of Sobolev regularity and, once again, to delve into the concept of regularity itself. In this new framework of thinking, we can mention a paper whose first version dates back to 2014 and was revised in October 2017 by **Francis Bach** (ENS/Inria): *"Breaking the curse of dimensionality with Convex Neural Networks"*[36]. Let us recall that in this type of problem, there are three facets to consider:

— **Approximation**: If we have an oracle that provides the weights, what is the error in approximating the function $f(x)$?

— **Optimization**: What algorithm can be used to obtain the weights?

— **Estimation**: We have only a "reasonable" number of N examples, how do we proceed?

That said, we will focus on the first problem, which is Approximation, because it ultimately conditions the rest. What are the assumptions about f that we can make for the Approximation problem to have a reasonable solution? F. Bach has compiled a list of "classic" interesting assumptions, of which there are three.

In the first assumption (Dimensionality Reduction): we consider the set Ω such that

$$x \in \Omega \subset \mathbb{R}^d \quad \text{with } \dim(\Omega) = s \ll d \tag{126}$$

In other words, Ω is a manifold in $\mathbb{R}^d$. The simplest version is that of a linear manifold, so

$$f(x) = g(\mathbf{W}^T x) \quad \text{with} \quad \text{rank}(\mathbf{W}) = s < d \tag{127}$$

36. source: 2014arXiv1412.8690B, see a presentation https://www.di.ens.fr/~{}fbach/fbach_cifar_2014.pdf

meaning that $\mathbf{W}$ is a $d \times s$ matrix. Through this, the number of relevant variables is actually s. In this case, as we outlined in the previous section, the Universality and Maiorov theorems apply with the replacement of d by s (*note: this resembles feature reduction*):

$$\epsilon \approx K^{-(m/(s-1))} \tag{128}$$

There are types of problems that naturally lend themselves to this dimensionality reduction: e.g., measurements at different points on an articulated arm because the joints introduce structural constraints.

The second assumption uses "Interaction Separability". This is often found in physical problems. For example, in a probabilistic case where we study Markov Models:

$$x \in \mathbb{R}^d, \quad f(x) = \sum_{j=1}^{J} f_j(x_i; i \in I_j) \tag{129}$$

meaning that f decomposes into a sum of functions, each of which depends on only a small number of variables. One can imagine an image in which local operations are performed that only involve a small number of pixels for each operation. In physics, this type of decomposition is found when long-range interactions can be neglected. For a Markov probability problem, it's as if we only need to know local conditional probabilities. So, this separability assumption is crucial and is found in many domains. It helps break the curse of dimensionality.

If all the I_j are of dimension s, we have J problems of dimension s. In this case, the number of "neurons" is on the order of

$$\epsilon \approx J \times K^{-(m/s)} \Leftrightarrow K \approx (\epsilon/J)^{-m/s} \tag{130}$$

This is a situation that occurs in many problems. What Francis Bach looks at is the case where

$$f(x) = \sum_{j=1}^{J} f_j(\langle x, w_j \rangle) \tag{131}$$

meaning that we project the d variables onto a dot product, so $s = 1$, and in this case

$$\epsilon \approx J \times K^{-(m)} \Leftrightarrow K \approx (\epsilon/J)^{-m} \tag{132}$$

The third assumption uses "Sparsity or Parsimony". In this context[37], we ask whether we can reduce the number of descriptors for the function $f(x)$. We define a dictionary $\mathcal{D} = \{g_m\}_{m\leq M}$ with potentially many features/patterns/descriptors g_m; however, to obtain an approximation $\tilde{f}$ of f, we can take only **a small number** of them:

$$\tilde{f}(x) = \sum_{m\in I} \alpha_k g_k(x), \quad \|f - \tilde{f}\| \leq \epsilon \text{ with } |I| = K \text{ such that } \epsilon \sim K^{-\alpha} \tag{133}$$

In other words, we take the smallest number of descriptors (K), and at the same time, we want an approximation error that decreases rapidly as we increase this number.

Recall that in Fourier analysis, we had the same approach, i.e., we do not keep all frequencies but restrict ourselves to low frequencies $|w| < C$ (cf. Perceptron regularization). However, the constant C depends solely on the nature of the differentiability of f through m, but it does not fundamentally depend on the function f itself: two functions that are three times differentiable will have nearly the same value of C.

In the sparsity assumption, we allow ourselves to **adapt to the function** f, especially I depends on f. Therefore, we introduce **adaptability** and, above all, the coefficients α_k will depend on f, i.e., we naturally introduce **non-linearity**. Adaptability can be understood quite easily, as in the case of Figure 52, where adaptive sampling is more effective than regular sampling in capturing the abrupt variability of the function itself.

A result by **A. R. Barron** was proposed in 1991: instead of imposing the Sobolev constraint (Eq. 119), a weaker assumption is made:

$$\sum_w |w|\ |\hat{f}(w)| < \infty \tag{134}$$

then the number of terms (neurons) will decrease/increase as follows:

$$\boxed{K \propto \frac{1}{\epsilon}} \tag{135}$$

meaning that **the curse of dimensionality has been removed**! This result is cited many times in machine learning books. This result seems miraculous because let's recall the

37. Note: This notion was also addressed in 2018: Wavelet Analysis.

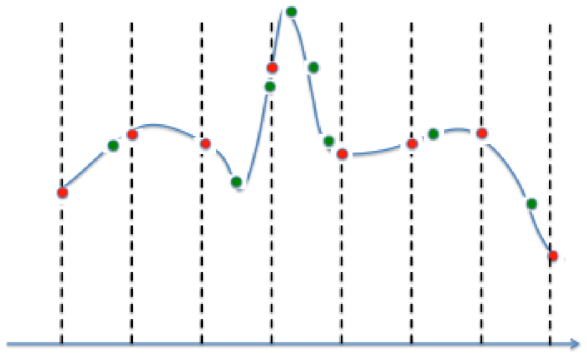


FIGURE 52 – Example illustrating the difference between regular sampling (red points) regardless of the function to be processed and adaptive sampling that takes into account the abrupt variations of the function to be approximated (green points).

Sobolev assumption:

$$\sum_w |w|^{2m} |\hat{f}(w)|^2 < \infty \tag{136}$$

at first glance, it is not clear what has changed. However, **it is not at all the same thing because remember that** $w \in \mathbb{Z}^d$ (see the Universality Theorem), and there is a notion of "hidden" sparsity that is almost never satisfied in Fourier analysis. Let's see how A.R. Barron's new assumption works. We will make it even simpler to show the connection with sparsity. Recall that if $f \in L^2$ and thus has finite energy, then

$$\|f\|_{L2}^2 = \sum_w |\hat{f}(w)|^2 < \infty \tag{137}$$

This is a strong constraint.

Theorem 6 *If we assume*

$$A = \sum_{w \in \mathbb{Z}^d} |\hat{f}(w)| < \infty$$

then

$$\exists \; \{w_k\}_{k \leq K} \text{ with } K = (A/\epsilon)^2 \; / \; f_K(x) = \sum_{k=1}^{K} \hat{f}(w_k) e^{-iw_k x}, \; \|f - f_K\|^2 \leq \epsilon^2 \quad (138)$$

The idea is to show that we get rid of the exponent d. In fact, the constraint is an **L1 norm**, and as soon as this norm is used, there is **sparsity**[38]. For the proof, we order the Fourier coefficients in decreasing order

$$\mathcal{F} = \{\hat{f}(w_k) / |\hat{f}(w_k)| \geq \hat{f}(w_{k+1})\} \quad (139)$$

We then use the following result:

Lemma 2

$$A = \sum_{w \in \mathbb{Z}^d} |\hat{f}(w)| \Rightarrow \hat{f}(w_k) \in \mathcal{F}, \quad |\hat{f}(w_k)| \leq \frac{A}{k} \quad (140)$$

If there is a mechanism that allows us to obtain an approximation by taking only frequencies belonging to a set I, then the error comes from the frequencies that are not taken into account:

$$f_I = \sum_{w \in I} \tilde{f}(w) e^{-iw.x} \Rightarrow \|f - f_I\|^2 = \sum_{w \notin I} |\tilde{f}(w)|^2 \quad (141)$$

Therefore, now to select the right frequencies, we will take those for which the Fourier coefficients are the largest, so

$$f_K = \sum_{k=1}^{K} \tilde{f}(w_k) e^{-iw_k.x} \quad (142)$$

meaning that $I = \mathcal{F}_K \subset \mathcal{F}$ for which we constrain $k \leq K$. Then

$$\|f - f_K\|^2 = \sum_{k > K} |\tilde{f}(w_k)|^2 \leq \sum_{k > K} \frac{A^2}{k^2} \leq A^2 \int_K^\infty \frac{dx}{x^2} = \frac{A^2}{K} \quad (143)$$

38. Note: See the types of L1, L2 regularization in the 2018 course: Classification/Regression in high dimension

and thus if we take $K = A^2/\epsilon^2$ then

$$\|f - f_K\|^2 \le \epsilon^2 \tag{144}$$

This is the result of the theorem. Therefore, sparse approximation consists of selecting the largest Fourier coefficients, and if the coefficients decay fast enough, the constraint on K is independent of the dimension d. The remaining task is to prove Lemma 2 concerning the L1 norm that determines the decay of the coefficients. We have (where w_k are the frequencies with ordered Fourier coefficients, cf. $\mathcal{F}$), $\forall P$:

$$\begin{aligned} A = \sum_{w\in\mathbb{Z}^d} |\hat{f}(w)| = \sum_{k=1}^{P} |\tilde{f}(w_k)| + \sum_{k>P} |\tilde{f}(w_k)| \ge \sum_{k=1}^{P} |\tilde{f}(w_k)| \\ \ge P \times |\tilde{f}(w_P)| \end{aligned} \tag{145}$$

So, we indeed have:

$$\forall P \quad |\tilde{f}(w_P)| \le \frac{A}{P} \tag{146}$$

which is the result of the lemma.

This result is more general than the particular case of the Fourier transform; as long as the coefficients in an orthogonal basis are controlled by an L1 norm. In fact, A.R. Barron uses the condition:

$$\sum_w |w|\ |\hat{f}(w)| < \infty \Rightarrow K \propto 1/\epsilon \tag{147}$$

and not $1/\epsilon^2$ as above, but the idea is the same.

However, why does Baron's miraculous theorem **not apply** to real problems? For the simple reason that, with very few exceptions, **functions $f(x)$ (in image classification, sound, etc.) are not sparse in Fourier space**. So, it's a very nice theorem but it doesn't apply. In summary, in the 2000s, with all the forms of regularity considered, the result of increasing the number of layers is not very clear: will it change the outcome?

7.4.3 Increasing the Number of Layers: It's Better!

There is a great paper that shows that, on the contrary, increasing the number of layers is better. It's an article by **Ronen Eldan and Ohad Shamir** (2015-16): "The power of depth of feedforward networks" [39]. The authors show that **there exists a simple radial function of $\mathbb{R}^d$ expressible with a 3-layer network (2 hidden layers) that cannot be expressed with a 2-layer network (1 hidden layer) without using an exponential number of neurons**. This function is the Fourier transform of the unit-volume ball of radius R_d in d-dimension:

$$\phi(x) = \left(\frac{R_d}{||x||}\right)^{d/2} J_{d/2}(2\pi R_d ||x||) \tag{148}$$

So, through this counterexample to the universality theorem (note that this doesn't mean the theorem is false, you have to be careful about the assumptions): **it's better to increase the number of layers than to increase the number of neurons in a smaller network**. To build their function, Eldan and Shamir use the fact that to approximate their function, you need **a lot of high-frequency power**, which kills the $|w| < C$ approximation in the proof of the 1 hidden layer network. And they use the **high-frequency rotation symmetry** so that adding one more hidden layer works very well.

So, the result is interesting for the idea that multi-layer networks are better. However, the functions in their "demonstration" are not realistic.

8. Neural Network Optimization

8.1 Introduction

Let's recall the diagram in Figure 1 representing the parameterized algorithm that maps x to $\tilde{y}$, the output of the neural network. In fact, $\tilde{y}$ depends on the parameterization θ, and we denote it as $\tilde{y}_\theta$. In the case of **Classification**, we have input-output pairs (x, y) with y an **index**, which is quite different from the case of **Regression** where $y \in \mathbb{R}^p$. In the case of **indices**, there is no natural topology; one can index in multiple ways. Thus,

39. https://arxiv.org/abs/1512.03965

the notion of continuity of the underlying function $f(x)$ that $\tilde{y}_\theta$ must approximate is not at all clear!

In terms of optimization, we define the notion of risk (see the 2018 course):

$$r(y,\tilde{y}) = \left| \begin{array}{ll} 0 & y=\tilde{y} \\ & \\ 1 & y\neq\tilde{y} \end{array} \right. \tag{149}$$

The total error on an empirical database used for learning, for example, is defined from the risk:

$$\tilde{R}(\theta) = \frac{1}{n}\sum_{i=1}^{n} r(f_\theta(x_i), y_i) \tag{150}$$

We hope that this empirical risk converges to the average risk as n tends to ∞:

$$R(\theta) = E_{(x,y)}[r(y, f_\theta(x))] \tag{151}$$

What's the problem? Certainly, we have gradient descent algorithms to minimize the risk, but that assumes that the function $\tilde{R}(\theta)$ is **differentiable**. **However, in classification, the risk $r(y,\tilde{y})$ is not differentiable at all, and this adds to the fact that y are indices.** The approach to this problem is to find quantities that are regular and can be estimated with regular f_θ that do not depend on too many parameters. Then, we define a risk that can be differentiated with respect to θ.

8.2 Bayesian Approach and Maximum Likelihood Principle

8.2.1 Transforming the Problem via Bayes

So, we have the average risk introduced in the previous section:

$$R = E_{(x,y)}[r(y, \tilde{y}(x))] \tag{152}$$

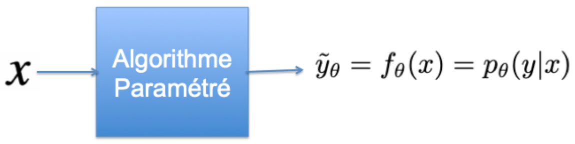


FIGURE 53 – The neural network viewed as a "machine" that approximates conditional probability densities (normalized).

One can ask what is the best approximation $\tilde{y}(x)$ of y? To answer this question, we can use conditional probabilities to write that

$$R = E_x \left(\sum_y r(y, \tilde{y}(x)) \; p(y|x) \right) \tag{153}$$

with $\sum_y p(y|x) = 1$. Now, $y = \{0, 1\}$, and we want to minimize the risk R, so we would like $r(y, \tilde{y}(x)) = 0$ in the case where the probability $p(y|x)$ is maximal. Therefore,

$$\boxed{r(y, \tilde{y}(x)) = 0 \text{ (meaning : } y = \tilde{y}) \quad \text{when} \quad p(\tilde{y}|x) = \max_y p(y|x)} \tag{154}$$

This is the **Bayesian Classifier**: $\tilde{y}(x)$ is the **most likely class** given the observation x. This implies that we choose the class that maximizes $p(y|x)$, and as a result, $\tilde{y}(x)$ **minimizes the error on average**. This is the **ideal classifier**, but to implement it, we need to know the probability distribution $p(y|x)$. **Therefore, we transform a problem of approximating y (indices) into the approximation of $p(y|x)$ continuous functions (for all y).** Note that if we have 10 possible values of y (10 classes), then there are 10 functions of x to approximate: $p(y|x) = g_y(x)$. These $g_y(x)$ are **well-regularized and well-structured functions.**

The Neural Network is viewed from this perspective as a "machine" that approximates conditional probability densities (normalized). The emerging schema is shown in Figure 53.

8.2.2 Maximum Likelihood

So, we want $p_\theta(y|x)$ to be an approximation of the true probability distribution $p(y|x)$. Therefore, we need a metric that gives us an estimate of the approximation error.

In this context, the **maximum likelihood** will appear as a particular case of minimizing the error between $p_\theta(y|x)$ and $p(y|x)$ using the specific distance of the **Kullback-Leibler divergence**.

The framework is as follows: given a family of Data $= \{x_i, y_i\}_{i \leq n}$, how do we approximate $p(y|x)$? If the sample family is drawn from $p_\theta(y|x)$, then the principle of maximum likelihood is to choose $\theta = \theta^*$ such that the probability of the data is maximized for $p_{\theta^*}(y|x)$. It is assumed that the data are all **independent**.

The probability of obtaining y_i given x_i according to p_θ is by definition $p_\theta(y_i|x_i)$. If the data are all **independent**, then the probability of obtaining the n samples in the family is simply the product of individual probabilities, i.e.,

$$\prod_{i=1}^{n} p_\theta(y_i|x_i) \tag{155}$$

and thus,

$$\begin{aligned}\theta^* &= \underset{\theta}{\text{argmax}} \left[\prod_{i=1}^{n} p_\theta(y_i|x_i)\right] \\ &= \underset{\theta}{\text{argmax}} \left[\log\left(\prod_{i=1}^{n} p_\theta(y_i|x_i)\right)\right] \\ &= \underset{\theta}{\text{argmax}} \left[\frac{1}{n}\sum_{i=1}^{n} \log p_\theta(y_i|x_i)\right] \end{aligned} \tag{156}$$

(*Note that the normalization factor is not important*).

Now, the average of "log" can be expressed as the expectation of "log" over the empirical distribution of the sample family, and thus, the maximum likelihood principle is translated into

$$\boxed{\theta^* = \underset{\theta}{\text{argmax}} \underset{\{x,y\}\sim\text{Data}}{E} (\log p_\theta(y|x))} \tag{157}$$

This Likelihood Principle can be seen as the minimization of a "distance": the Kullback-Leibler divergence.

8.2.3 Kullback-Leibler Divergence

The Kullback-Leibler divergence[40] is defined as

$$D_{KL}(p||q) = E_p\left(\log\frac{p}{q}\right) = \int p(x)\log\frac{p(x)}{q(x)}\,dx \tag{158}$$

It is not a strict distance, but it has important properties:

— $D_{KL}(p||q) \geq 0$, which can be shown by noting that $\forall x > 0$, $\log x \leq x - 1$ (equality for $x = 1$);

— It is not symmetric (so it is not a distance);

— One way to understand it is to imagine symbols produced by $p(x)$ and wanting to code these symbols optimally. However, you are given the distribution $q(x)$ instead of $p(x)$, so you will have coding inefficiency: $D_{KL}(p||q)$ is precisely this inefficiency. Why? Because the optimal Shannon code associated with $p(x)$ is of size $-\log(p(x))$, but you have $q(x)$ instead, so you get the optimal code $-\log(q(x))$, and the difference between the two reveals $\log(p/q)$, which on average yields $p\log(p/q)$.

— There are many other interpretations of this "distance" that appear everywhere in information theory (including machine learning).

— If $D_{KL}(p||q) = 0$, then $p = q$ (this is the particular case of the first point above).

Therefore, it makes sense to use this divergence to optimize a neural network that attempts to approximate $p(y|x)$ with $p_\theta(y|x)$.

Theorem 7 *Maximum likelihood will minimize*

$$D_{KL}(p_{\text{Data}}(y|x)||p_\theta(y|x))$$

The proof is quite simple; it is sufficient to write the value of the divergence as

40. NB: The Kullback-Leibler divergence or K-L divergence, also known as relative entropy, is named after Solomon Kullback and Richard Leibler, two American cryptanalysts from the NSA who invented this concept in the 1950s.

follows:

$$\begin{aligned} D_{KL}(p_{\text{Data}}||p_\theta) &= E_{p_{\text{Data}}}\left(\log \frac{p_{\text{Data}}}{p_\theta}\right) \\ &= E_{p_{\text{Data}}}\left(\log p_{\text{Data}}(y|x)\right) - E_{p_{\text{Data}}}\left(\log p_\theta(y|x)\right) \end{aligned} \tag{159}$$

and we want to minimize this with respect to θ. Now, only the second term depends on θ, and minimizing it is equivalent to maximizing its opposite, which is exactly the definition of maximum likelihood.

Therefore, choosing the parameter θ that maximizes likelihood is equivalent to finding the probability distribution that best approximates the empirical distribution (i.e., obtained from the data itself) according to the Kullback-Leibler divergence metric.

8.2.4 Relation with Bayesian Models

8.2.4.1 A Preliminary Comment

In the literature on machine learning, there are consistently two viewpoints:

— **A purely deterministic viewpoint** that starts from the observation that the network provides the response $y = \tilde{f}(x)$, which should best approximate the function $f(x)$. This falls into the domain of function approximation, which is entirely deterministic (i.e., no probability involved).

— **A purely probabilistic and inherently Bayesian viewpoint** that tries to approximate probability densities in this context.

In fact, **these two viewpoints are equivalent**, and researchers often switch between them within the same publication to identify the right "object" to estimate. For example, in the case of regression with "well-behaved" functions, there is no need for the probabilistic viewpoint. However, in classification, where nothing is well-behaved initially, the use of probability distributions is a way to reformulate the problem in a regular framework.

8.2.4.2 Bayesian Approach vs. Deterministic Approach

In the Bayesian approach, we have **data**, Data $= \{x_i, y_i\}_{i\leq n}$, and parameters θ that model these data. Furthermore, we would like to obtain θ^* that maximizes the probability of θ given the dataset:

$$\theta^* = \underset{\theta}{\text{argmax}}\ p(\theta|\text{Data}) \tag{160}$$

This is the **maximum a posteriori** (MAP) estimation. According to Bayes' theorem:

$$p(\theta|\text{Data}) = \frac{p(\text{Data}|\theta)\ \pi(\theta)}{p(\text{Data})} \tag{161}$$

with $p(\text{Data}|\theta) \equiv \mathcal{L}(\theta)$ as the **likelihood**, $\pi(\theta)$ as the **prior** on θ, and $p(\text{Data})$ as the probability of obtaining the data. We want to calculate the max with respect to θ, so only the numerator matters.

If we have no *a priori* information about θ, then $\pi(\theta) = \text{constant}$, and therefore, the value of θ is such that:

$$\theta^* = \underset{\theta}{\text{argmax}}\ p(\text{Data}|\theta) \tag{162}$$

which is nothing but **the maximum likelihood principle that maximizes the likelihood**. The Bayesian formulation and the maximum likelihood principle coincide only if $\pi(\theta) = \text{constant}$.

However, if $\pi(\theta) \neq \text{constant}$, then maximizing the probability is equivalent to maximizing the logarithm of the probability. Thus:

$$\boxed{\theta^* = \underset{\theta}{\text{argmax}}\left[\log p(\text{Data}|\theta) + \log \pi(\theta)\right]} \tag{163}$$

The term $\log \pi(\theta)$ is a **penalty** that can also be found in neural networks when defining the cost function. In fact, we can write:

$$\tilde{R}(\theta) = \frac{1}{n}\sum_{i=1}^{n} r(f_\theta(x_i), y_i) + C(\theta) \tag{164}$$

where the function $C(\theta)$ arises from the *a priori* knowledge about θ and is formulated as a constraint, such as using L2 regularization on the weights. In other words, imposing that the L2 norm on the parameter weights is not too large **is a prior information (cf. $\pi(\theta) \neq$ constant) to constrain the value of** θ.

Thus, **we can see that expressing a cost function in a purely deterministic domain with regularization $C(\theta)$ can also be seen as a purely Bayesian version with prior probability on** θ.

8.2.4.3 Regression Case

In fact, we come back to the same ideas as before. In this case, we have $y = f(x)$ that we want to approximate with $y_\theta = f_\theta(x)$, and we want to minimize the empirical quadratic risk in the mean:

$$\min_\theta \sum_{i=1}^{n} |y_i - f_\theta(x_i)|^2 \tag{165}$$

What is the connection with Maximum Likelihood? The network calculates $f_\theta(x)$, and we would like to associate it with a probability distribution $p_\theta(y|x)$. In general, we use a Gaussian centered on the computed value $f_\theta(x)$ with an error σ:

$$p_\theta(y|x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y-f_\theta(x))^2}{2\sigma^2}} \tag{166}$$

If we now apply maximum likelihood, which is equivalent to minimizing the Kullback-Leibler divergence:

$$\begin{aligned}\min_\theta\ D_{KL}(p_{\text{Data}}||p_\theta) &= -\min_\theta E_{p_{\text{Data}}}(\log p_\theta(y|x)) \\ &= \min_\theta\ E_{p_{\text{Data}}}((y - f_\theta(x))^2) \\ &= \min_\theta\ \sum_{i=1}^{n} |y_i - f_\theta(x_i)|^2 \end{aligned} \tag{167}$$

So, minimizing a quadratic error (cost) is also equivalent to saying that we take the maximum of a Gaussian likelihood centered on the network's output.

Both approaches are therefore equivalent, and in this case, they do not bring much difference. However, in classification, where there is no *a priori* metric, we have seen how the Kullback-Leibler divergence (or Maximum Likelihood) provides the solution.

8.3 Implementation for a Neural Network (Classification)

8.3.1 Introduction

The technique is quite generic and applies to a wide class of classifiers. Recall Figure 53 with θ representing all the coefficients of the linear matrices W_k and all the biases b_k

(here, we have one layer). However, one property of probabilities $p_\theta(y|x)$ is that:

$$\sum_y p_\theta(y|x) = 1 \tag{168}$$

But the network's outputs have no reason to be normalized, **so we must add a normalization step**. Once this step is performed, the best $\tilde{y}$, meaning the best class (for a given x), is the one that satisfies:

$$\tilde{y} = \underset{y}{\operatorname{argmax}} \left[p_{\theta^*}(y|x)\right] \tag{169}$$

This is an entirely Bayesian approach.

8.3.2 Introduction of Softmax

Now, how can we achieve normalization? The `softmax` function will associate with the output, let's call it $z_y(x)$, which is the numerical value z for index y given x, the value:

$$z_y(x) \xrightarrow{\text{softmax}} \frac{e^{z_y(x)}}{\sum_{y'} e^{z_{y'}(x)}} = p_\theta(y|x) \tag{170}$$

We use this definition and not the simple expression $z_y / \sum_{y'} z_{y'}$ because the z_y are not necessarily positive. Moreover, if there is a dominant value, the exponential reinforces this state. There is another advantage that we will examine now.

According to the previous section, we want to minimize the D_{KL} divergence, which is expressed as:

$$\sum_i -\log p_\theta(y_i|x_i) \tag{171}$$

where the expression of the probability $p_\theta(y_i|x)$ is given by the `softmax`. Therefore, we want to **minimize** (*in fact, z_y depends on θ but the notation becomes heavy*):

$$L(\theta) = \sum_i -\log\left(\frac{e^{z_{y_i}(x_i)}}{\sum_{y'} e^{z_{y'}(x_i)}}\right) = -\sum_i \left[z_{y_i}(x_i) - \log\left(\sum_{y'} e^{z_{y'}(x_i)}\right)\right] \tag{172}$$

This function $L(\theta)$, for a given sample i, directly depends on the network's output thanks to the `softmax`, and on a normalization term. Having $L(\theta)$ directly linked to $z_{y_i}(x_i)$ will allow efficient conditioning of the minimization problem. Indeed, we want to minimize $L(\theta)$, so we want to maximize $z_{y_i}(x_i)$, i.e., maximize the probability of the response $y_\theta(x_i) = y_i$. The other term looks like the maximum of $z_{y'}(x_i)$, especially if this maximum is well separated from the other values:

$$\log\left(\sum_{y'} e^{z_{y'}(x_i)}\right) \approx \max_{y'}\left[z_{y'}(x_i)\right] \tag{173}$$

Thus, $L(\theta)$ boils down to comparing $z_{y_i}(x_i)$ and $\max\limits_{y'}\left[z_{y'}(x_i)\right]$:

$$L(\theta) \approx -\sum_i \left(z_{y_i}(x_i) - \max_{y'}\left[z_{y'}(x_i)\right]\right) \tag{174}$$

Therefore, during an iteration of minimization on θ, if the maximum of $z_{y'}(x_i)$ is achieved for $y' = y_i$, then the difference is 0, and we have won because there is no need to update θ. The `softmax` allows us to do this with a differentiable function.

8.3.3 Optimization: Special Case of Classification by Logistic Regression

Logistic regression (in fact, it is classification) can be viewed as a neural network with only one unhidden layer whose response z is a linear function of the input x (no non-linear activation function):

$$z = \mathbf{W}x + b \tag{175}$$

If we take the yth row of this matrix, it represents the response $z_y(x)$ as follows:

$$z_y(x) = \langle x, w_y\rangle + b \tag{176}$$

with y an integer as if it were a class index. Furthermore, noting that $\theta = \mathbf{W}$, we want to minimize the following cost function:

$$L(\mathbf{W}) = -\sum_i \left[(\langle x_i, w_y\rangle + b) - \log\left(\sum_{y'} e^{\langle x, w_{y'}\rangle + b}\right)\right] \tag{177}$$

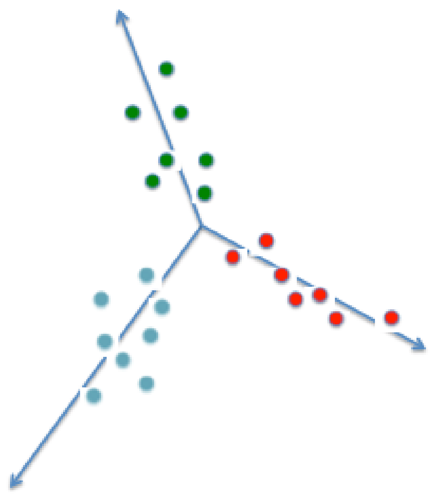


FIGURE 54 – Schematic representation of the vectors w around which the data aggregates.

The geometric interpretation is as follows: as a first remark, w_y has the same dimension as x, and they both evolve in the same space. So, the w_y each represent a direction in this space. The classifier will try to find the direction w_y that coincides for class y_i with the direction w_{y_i} around which the x_i aggregate (see Figure 54). This is the effect of the first term; the other term in the $L(\mathbf{W})$ function will constrain the direction w_y to not be in the direction of the other classes.

What happens if we have an error, i.e., an "outlier" for sample i? For example, if we have 2 classes $\{-1, 1\}$, then $w_1 = -w_{-1}$ (see Figure 55), and suppose that sample i is classified as -1 instead of 1. The penalty is equal to:

$$-(z_{y_i}(x_i) - \max_{y'} [z_{y'}(x_i)]) = -(z_1(x_i) - z_{-1}(x_i)) = -\langle x_i, w_1 \rangle + \langle x_i, w_{-1} \rangle = 2\langle x_i, w_{-1} \rangle \quad (178)$$

Now, x_i is located in the direction of w_{-1} relative to the separating plane. **So, we penalize by (2 times) the distance it takes to bring it back to the correct side of the plane** (i.e., we take its symmetry with respect to the plane). This is very similar to what the **SVM** algorithm does for the margin term that serves as regularization (see section 5.2.4).

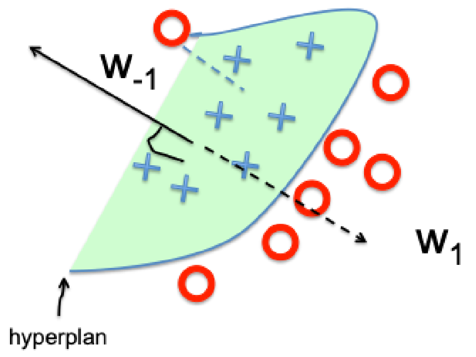


FIGURE 55 – Outlier in the case of 2 classes.

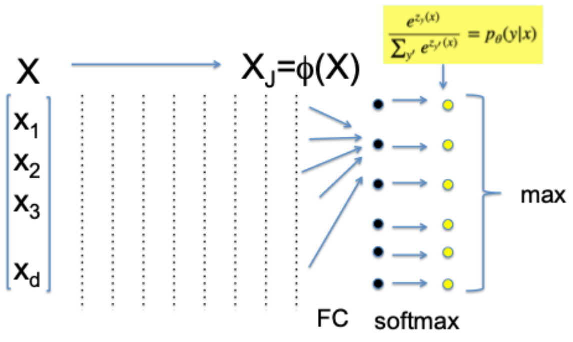

$\frac{e^{z_y(x)}}{\sum_{y'} e^{z_{y'}(x)}} = p_\theta(y|x)$
X
X_J=φ(X)
x1
x2
x3
xd
max
FC
softmax

FIGURE 56 – Schematic representation of the different parts of an MLP.

8.3.4 For a Multi-Layer Perceptron (MLP) Neural Network

In the case of an MLP, there are several hidden layers for which the transformation between two successive layers involves the following relation:

$$x_j = \sigma\left(\mathbf{W}_j x_{j-1} + b_j\right) \tag{179}$$

up to layer J where we obtain the representation $x_J = \Phi(x)$. Then, we aggregate using logistic regression to obtain a classification vector through the `softmax`. Finally, we can, for example, take the `max` to get $\tilde{y}$ (see Figure 56).

We optimize **jointly** the data representation ($\Phi(x)$) and the classifier (logistic). The prior information is contained in the architecture. Note that a large number of samples is also required to learn the representation. In the next section, we will address the algorithmic part of MLPs.

9. Optimization Algorithms for MLPs

We will discuss the "Backprop" algorithm for gradient backpropagation and stochastic gradient descent. Later, we will delve into the proofs.

9.1 Gradient Computation in MLPs (Back-propagation)

See the 1986 article by Rumelhart, Hinton, Williams [41], which exploits gradients of composite functions—a "simple" idea that proves to be highly effective in the context of MLPs [42].

9.1.1 Forward & Backward Flow

The multi-layer network can be represented as a stack of layers with inputs, outputs, and operations F (see Figure 57). The first layer of neurons performs the operation (*note that* $\mathbf{W}$ *often implicitly includes the bias with a convention of adding a 1 for* x_i):

$$x_1 = F_1(x_0, \mathbf{W}_1) = \sigma(\mathbf{W}_1 x_0 + b_1) \tag{180}$$

and so on for F_2, F_3, etc., up to F_J which produces the output x_J, which is compared by a loss function (*loss*) to the output y of a labeled sample. The set of network parameters (the $\mathbf{W}_i, b_i$) is denoted as θ, some of which can potentially be shared by multiple F_i.

41. Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986) Learning representations by back-propagating errors. Nature, 323, 533–536.

42. NDJE: I pointed out in 2017 that Werbos discussed this in 1974 and, in 1982, applied it specifically to neural networks. LeCun and Parker also published related ideas in 1985. However, the ideas of error minimization through gradient descent date back to Cauchy in 1847 and Hadamard in 1908. So, determining who first invented the Back-Propagation algorithm is a story in itself.

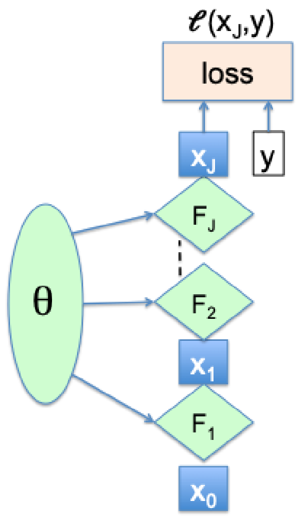


FIGURE 57 – Schematic representation of the "forward" flow in which we follow a sample x as it passes successively through the different layers of the network, to be compared to y in a loss function. The network parameters (denoted θ) come into play in describing the operations F at each layer.

Thus, the global average loss function that depends on θ, denoted $\ell(\theta)$, is calculated over a batch of labeled samples $\{x_0^i, y^i\}_{i\leq N}$ as follows:

$$\ell(\theta) = \frac{1}{N}\sum_{i=1}^{N} \ell(x_J^i, y^i) \tag{181}$$

where θ comes into play in the calculation of x_J^i. To find the minimum of this cost function, we use *gradient descent*, which, at step n, defines a set of parameters denoted θ_n, and the value $n+1$ is derived by adding a contribution directed opposite to the gradient of $\ell(\theta)$ calculated with θ_n. Thus,

$$\begin{aligned}\theta_{n+1} &= \theta_n - \alpha \nabla_\theta \ell(\theta) \\ &= \theta_n - \alpha \frac{1}{N}\sum_{i=1}^{N} \nabla_\theta \ell(x_J^i, y^i)\end{aligned} \tag{182}$$

Therefore, the problem is to calculate $\nabla_\theta \ell(x_J^i, y^i)$ for any input $x_0^i \equiv x^i$. The principle is as follows: the output of the j-th layer is calculated as

$$x_j = F_j(x_{j-1}, \mathbf{W}_j) \tag{183}$$

If we now know the variation of the *loss* with respect to this output, i.e., we know $\nabla_{x_j}\ell$, then

$$\nabla_{\mathbf{W}_j}\ell = \nabla_{x_j}\ell \times \nabla_{\mathbf{W}_j} F_j \tag{184}$$

So, it is interesting to calculate not only $\nabla_{\mathbf{W}_j}\ell$ but also $\nabla_{x_j}\ell$. This is visualized in the diagram in Figure 58, where at step j, we distinguish between an "upward" or **Forward** flow of inputs/outputs x_j and a "downward" or **Backward** flow of inputs/outputs $\nabla_{x_j}\ell$, i.e., **the backpropagation of error variations**.

9.1.2 Initialization: Calculating $\nabla_{x_J}\ell$?

9.1.2.1 In the Case of Regression

The expression for the loss is typically given by

$$\ell(x_J, y) = \frac{1}{2}(x_J - y)^2 \tag{185}$$

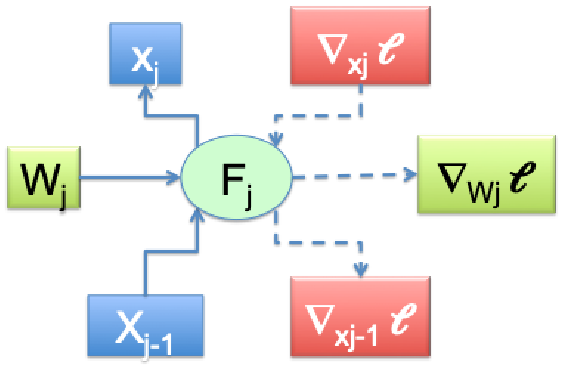


FIGURE 58 – Zoom on the inputs/outputs of layer j of the network 57: we distinguish between an "upward" or Forward flow of inputs/outputs x_j and a "downward" or Backward flow of inputs/outputs $\nabla_{x_j}\ell$, i.e., the backpropagation of error variations.

and it easily follows that

$$\partial_{x_J}\ell = x_J - y \tag{186}$$

9.1.2.2 In the Case of Classification

There is no *a priori* differentiability, and x_J is a K-dimensional vector where $x_J(k)$ signifies the probability of the k-th class estimated by a `softmax`; whereas y is a **one-hot vector** in which all elements are zero except for one class for which $y(k) = 1$. To address the differentiability issue, as seen in the previous lecture, we can use maximum likelihood or minimize the following loss:

$$\begin{aligned}\ell(x_J, y) &= -\sum_{k'=1}^{K} y(k')\log(\text{softmax}(x_J(k')))\\ &= -\log(\text{softmax}(x_J(k)))\\ &= -x_J(k) + \log\sum_{k'} e^{x_J(k')}\end{aligned} \tag{187}$$

where

$$\texttt{softmax}(x_J(k)) = \frac{e^{x_J(k)}}{\sum_{k'} e^{x_J(k')}} \tag{188}$$

The gradient is then easily calculated by separating the cases where we consider the

k component of x_J and y for which $y(k) = 1$, or the other components for which $y(k') = 0$:

$$\nabla_{x_J}\ell = \left| \begin{array}{rl} -1 + \frac{e^{x_J(k)}}{\sum_q e^{x_J(q)}} = \partial_{x_J(k')}\ell & \text{if} \quad k = k' \\ \frac{e^{x_J(k')}}{\sum_q e^{x_J(q)}} = \partial_{x_J(k')}\ell & \text{if} \quad k \neq k' \end{array} \right| = \mathtt{softmax}(x_J) - y \tag{189}$$

The expression is ultimately very simple thanks to the `softmax`. We realize that this gradient is the error between a "soft" (or smoothed) maximum of the x_J component that gives the probability of the corresponding class, and y, which is the "true" probability (i.e., the component with a value of 1 for the correct class). The big advantage of `softmax` is its differentiability.

9.1.3 Gradient Backpropagation

This involves the derivative of function composition:

— 1D: $h(x) = g(f(x))$ then $h'(x) = f'(x)g'(f(x))$ which can be reformulated in terms of variables $z = h(x)$ and $y = f(x)$:

$$\frac{dz}{dx} = \frac{dz}{dy}\,\frac{dy}{dx} \tag{190}$$

— In higher dimensions: $x = \{x_k\}_k$, $y = \{y_j\}_j$, then

$$\frac{\partial z}{\partial x_k} = \sum_j \frac{\partial z}{\partial y_j}\frac{\partial y_j}{\partial x_k} \tag{191}$$

If we introduce the Jacobian

$$\left(\frac{\partial y}{\partial x}\right) \equiv \begin{pmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_K} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_M}{\partial x_1} & \cdots & \frac{\partial y_M}{\partial x_K} \end{pmatrix} \tag{192}$$

then

$$\boxed{\nabla_x z = \left(\frac{\partial y}{\partial x}\right)^T . \nabla_y z} \tag{193}$$

So, when calculating $\nabla_{x_{j-1}}\ell$ and $\nabla_{\mathbf{W}_j}\ell$, we involve the Jacobians of x_j with respect to x_{j-1} and $\mathbf{W}_j$, which are

$$\nabla_{x_{j-1}}\ell = \left(\frac{\partial x_j}{\partial x_{j-1}}\right)^T . \nabla_{x_j}\ell = \left(\frac{\partial F_j(x_{j-1}, .)}{\partial x_{j-1}}\right)^T . \nabla_{x_j}\ell \tag{194}$$

$$\nabla_{\mathbf{W}_j}\ell = \left(\frac{\partial x_j}{\partial \mathbf{W}_j}\right)^T . \nabla_{x_j}\ell = \left(\frac{\partial F_j(., \mathbf{W}_j)}{\partial \mathbf{W}_j}\right)^T . \nabla_{x_j}\ell \tag{195}$$

So, we (retro)propagate the gradient (cf. we move from layer $j-1$ to layer j) by multiplying the Jacobians of the transformation in each "box"/"layer" (cf. the functions F_j) given in the case of a neural network by a transformation involving input-output by linear combination and a pointwise non-linearity like ReLU.

9.1.4 The Jacobians of F_j

We can decompose (*note the bias*)

$$F_j = F_j(x_{j-1}, \mathbf{W}_j) = \sigma(\mathbf{W}_j x_{j-1} + b_j) \tag{196}$$

into two steps (figure 59):

— A matrix step

$$x_{j-1} \xrightarrow{F_j^1} \mathbf{W}_j x_{j-1} \tag{197}$$

— A step of bias addition and application of non-linearity

$$x \xrightarrow{F_j^2} \sigma(x + b_j) \tag{198}$$

So, we can calculate the Jacobians of F^1 and F^2 quite easily. On one hand,

$$\frac{\partial F_j^1}{\partial \mathbf{W}_j} = x_{j-1} \quad , \quad \frac{\partial F_j^1}{\partial x_{j-1}} = \mathbf{W}_j^T \tag{199}$$

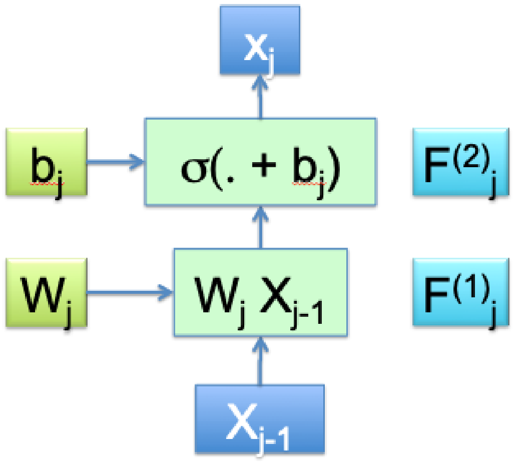


FIGURE 59 – Decomposition into two phases F^1 and F^2 of the transformation in a layer.

and on the other hand, for F_j^2, it is good to note that the variables are not mixed, so the Jacobian is diagonal and can be expressed as follows:

$$\frac{\partial F_j^2}{\partial x_{j-1}^1} = \mathbf{Diag}(\sigma'(x_{j-1}^1 + b_j)) \tag{200}$$

$$\frac{\partial F_j^2}{\partial b_j} = \mathbf{Diag}(\sigma'(x_{j-1}^1 + b_j)) \tag{201}$$

9.1.5 Graphical Representation of the Algorithm

We have already seen from figures 57, 58, and 59 the layered and sub-layered structure of this backpropagation algorithm, and we can generalize these graphs in the case of parameter sharing, for example. Graphs that can be modeled in this way have one constraint: **they do not include recursion**.

Otherwise, libraries like `Keras` or `pyTorch` allow you to define the gradients of different layers (if they are not of already known types) and apply gradient descent methods via *backprop*.

9.2 Convergence Study of GD

In the previous section, we developed the tools to perform *Gradient Descent (GD)*, but it is essential that GD converges to a minimum of the error on the training data, with the goal of achieving the smallest possible generalization error. Note that the latter depends on the class of functions that can be approximated with the chosen network architecture *a priori*, which influences the bias term of the error. So, in what situations does GD converge? What type of GD is needed for efficient convergence? Ultimately, the challenge lies in finding a network architecture that aligns with the GD method used and vice versa. Let's now consider two GD methods: the *Batch* version and the *Stochastic* version.

9.2.1 Batch or Stochastic GD

If we denote $z_i = (x_i, y_i)$, then the overall loss can be expressed in terms of the chosen variables as

$$\ell(\theta) = \frac{1}{N}\sum_{i=1}^{N} \ell(f_\theta(x_i), y_i) = \frac{1}{N}\sum_{i=1}^{N} \ell(\theta, z_i) \tag{202}$$

The $t+1$ step of GD can be written using the general method as

$$\begin{aligned}\theta_{t+1} &= \theta_t - \alpha \nabla_\theta \ell(\theta_t) \\ &= \theta_t - \alpha \left(\frac{1}{N}\sum_i \nabla_\theta \ell(\theta_t, z_i)\right) \qquad \text{(BGD)} \\ &= \theta_t - \alpha \langle \nabla_\theta \ell(\theta_t, z_i)\rangle_{i\le N}\end{aligned} \tag{203}$$

If it takes D operations to compute $\nabla_\theta \ell(\theta, z_i)$ by backpropagation, then to calculate θ_{t+1}, it roughly takes $N \times D$ operations.

The question that arose quite quickly is: is it necessary to use all N samples to calculate an instance of the average gradient? We can answer this by already considering only a subset M of the N samples, which is the **Batch** version. However, there is an even more drastic version that retains **only one sample at a time**, which is the **Stochastic** version. In this case, $i_t \in \{1, \dots, N\}$ is a sample chosen **at random**.

$$\theta_{t+1} = \theta_t - \alpha \nabla_\theta \ell(\theta_t, z_{i_t}) \qquad \text{(SGD)} \tag{204}$$

Randomness is important in this operation because then

$$E[\nabla_\theta \ell(\theta_t, z_{i_t})] = \sum_{i_t=1}^{N} \nabla_\theta \ell(\theta_t, z_{i_t}) \times P(i_t) = \frac{1}{N} \sum_{i_t=1}^{N} \nabla_\theta \ell(\theta_t, z_{i_t}) = \nabla_\theta \ell(\theta_t) \tag{205}$$

which highlights that $\nabla_\theta \ell(\theta_t, z_{i_t})$ is a **noisy gradient** that, on average, yields the global gradient of the loss.

The SGD method is much faster than the BGD method, but it converges much more slowly, especially towards the end when you are close to the minimum. Therefore, to make a decision, you need to be able to calculate the respective convergence rates. In the **convex case**, the **Batch method converges exponentially**, while the **Stochastic method converges in 1/N**.

It is interesting to note that the SGD method was described by Munro & Robbins in 1951[43], but it had never been put into practice due to its slow convergence, and especially because of the small size of the databases. The game changed as soon as 1) we tackled **non-convex problems**, and 2) we were able to use **substantial training datasets**.

9.2.2 Example of the Quadratic Function: Convergence of GD

Let's define the Hessian matrix (called the Hessian) of the loss as

$$H[\ell](\theta) \equiv \left(\frac{\partial^2 \ell}{\partial \theta_i \partial \theta_j} \right) \tag{206}$$

If τ is a unit vector in the θ space, then

$$\frac{\partial^2 \ell}{\partial \tau^2} = \tau^T \; H[\ell](\theta) \; \tau \tag{207}$$

In fact, this can be demonstrated by a brief reminder of "directional derivative". If $t \in \mathbb{R}$, $x \in \mathbb{R}^n$, and τ is a unit vector in $\mathbb{R}^n$, then let's define the function $g(t)$ as

$$g(t) = f(x + t\tau) = f(x_1 + t\tau_1, \ldots, x_n + t\tau_n) \tag{208}$$

43. Robbins, H. and Munro, S. "A Stochastic Approximation Method." Ann. Math. Stat. 22, 400-407, 1951.

When calculating the successive derivatives of $g(t)$, it follows that

$$g'(t) = \sum_{i=1}^{n} \tau_i \; \partial_{x_i} f(x_1 + t\tau_1, \ldots, x_n + t\tau_n) = \tau^T \; \nabla_x f(x + t\tau) \tag{209}$$

and

$$g''(t) = \sum_{i=1}^{n} \tau_i \left\{ \sum_{j=1}^{n} \tau_j \; \partial_{x_j} \partial_{x_i} f(x_1 + t\tau_1, \ldots, x_n + t\tau_n) \right\} = \tau^T \; \nabla_x^2 f(x + t\tau) \; \tau \tag{210}$$

Therefore, when taking the limit as $t \to 0$, we obtain the definition of directional derivatives of f in the direction τ at point x :

$$\boxed{\begin{aligned} \frac{\partial f}{\partial \tau} &= \nabla_\tau f(x) = \tau^T \; \nabla_x f(x) && (211) \\ \frac{\partial^2 f}{\partial \tau^2} &= \nabla_\tau^2 f(x) = \tau^T \; \nabla_x^2 f(x) \tau = \tau^T \; H[f](x) \tau && (212) \end{aligned}}$$

Now, if we perform a Taylor expansion of ℓ in the vicinity of a point θ_0 (*note: for notation simplicity, $H[\ell](\theta)$ is denoted as $H(\theta)$*), it follows that:

$$\ell(\theta) = \ell(\theta_0) + (\theta - \theta_0)^T \nabla \ell(\theta_0) + \frac{1}{2}(\theta - \theta_0)^T H(\theta_0)(\theta - \theta_0) + \ldots \tag{213}$$

If higher-order terms beyond 2 are negligible, the analysis becomes simpler. In fact, one iteration of gradient descent starting from θ_0 gives:

$$\theta = \theta_0 - \alpha \nabla \ell(\theta_0) \qquad (\alpha > 0) \tag{214}$$

and if we denote $\nabla \ell(\theta_0) = g$, then

$$\ell(\theta) = \ell(\theta_0 - \alpha g) = \ell(\theta_0) - \alpha ||g||^2 + \frac{\alpha^2}{2} g^T H(\theta_0) g \tag{215}$$

Therefore, we can observe that the GD step indeed reduces the function at the first order (as indicated by the '-' sign), and **the conditions for the existence of a minimum** can be written as:

$$||g|| = 0 \qquad \text{and} \qquad g^T H(\theta_0) g \geq 0 \tag{216}$$

This is the concept of **convexity**, and if we want this property to be universally defined, we require that these **2 conditions are met** $\forall\theta_0$, meaning that the **Hessian is a positive definite matrix** (i.e., all eigenvalues are positive). We can even calculate the **optimal step**, which is to determine the optimal value of α, namely:

$$\boxed{\alpha^* = \frac{||g||^2}{g^T H(\theta_0) g}} \tag{217}$$

but this requires being able to compute the Hessian.

Now, what is the order of magnitude of this optimal step? If the convexity condition is met, we can bound the eigenvalues of the Hessian. Furthermore, if we impose that the smallest eigenvalue is **non-zero**, then we obtain the following bound:

$$0 < \mu \leq \frac{g^T H(\theta_0) g}{||g||^2} \leq L \tag{218}$$

However, we can find ourselves in the unpleasant situation depicted in Figure 60, where we keep going back and forth in a narrow corridor. In other words, it must be realized that the gradient indicates the direction of "steepest ascent". Let's calculate the **convergence rate**. To do this, we want to estimate, step by step, how the distance between θ_t and θ^* (the minimum) behaves. So, during a step $t \to t+1$:

$$\theta_{t+1} = \theta_t - \alpha \nabla \ell(\theta_t) \tag{219}$$

where

$$\nabla \ell(\theta^*) = 0 = \nabla \ell(\theta_t) + H(\theta_t)(\theta^* - \theta^t) \tag{220}$$

thus

$$\theta_{t+1} = \theta_t - \alpha H(\theta_t)(\theta_t - \theta^*) \tag{221}$$

and

$$\theta_{t+1} - \theta^* = \theta_t - \theta^* - \alpha H(\theta_t)(\theta_t - \theta^*) = (1 - \alpha H(\theta_t))(\theta_t - \theta^*) \tag{222}$$

Therefore, through successive iterations, we get

$$\theta_{t+1} - \theta^* = (1 - \alpha H(\theta_t)) \dots (1 - \alpha H(\theta_0))(\theta_0 - \theta^*) \tag{223}$$

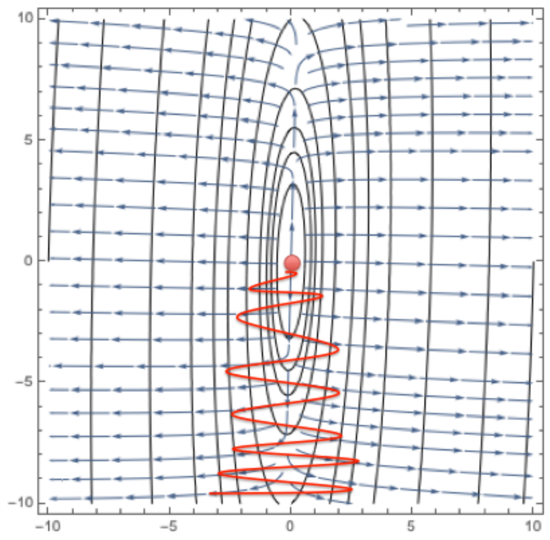


Figure 60 – A scenario where the function that needs to be minimized has a "narrow gutter" shape. Since the direction of the GD step is opposite to the local gradient, it is highly likely that starting from θ_0, the successive parameter values oscillate on either side of the gutter's edges, converging very slowly to the minimum.

(*note that here α is a constant of the process while we have seen earlier that it can be adapted for each step t*). For convergence to occur, the matrices $(1 - \alpha H(\theta_i))$ must be contractive, meaning that the norm of these matrices must be less than 1, i.e.,

$$\forall\theta \qquad ||1 - \alpha H(\theta)|| < 1 \tag{224}$$

and if we know that the eigenvalues of $H(\theta)$ are bounded:

$$\mu < ||H(\theta)|| < L \tag{225}$$

then we have at least the constraint that

$$\boxed{\alpha < \frac{1}{L}} \tag{226}$$

But the worst case is when the Hessian becomes very small:

$$\max_\theta ||1 - \alpha H(\theta)|| = 1 - \alpha\mu \tag{227}$$

then

$$||\theta_t - \theta^*|| \leq (1 - \alpha\mu)^t ||\theta_0 - \theta^*|| \tag{228}$$

However, we cannot guarantee the condition on α above, which limits the convergence rate to:

$$\boxed{||\theta_{t+1} - \theta^*|| \leq \left(1 - \frac{\mu}{L}\right)^t ||\theta_0 - \theta^*||} \tag{229}$$

meaning that **we would like the ratio μ/L to be as close to 1 as possible**, which constitutes **the conditioning of the Hessian**.

However, as is evident, even in the case illustrated in Figure 60, we are not necessarily going in the "right" direction at each step. Nevertheless, in this quadratic case, **if the Hessian is invertible**

$$\nabla\ell(\theta_t) = H(\theta_t)(\theta_t - \theta^*) \Rightarrow \theta_t - \theta^* = H(\theta_t)^{-1}\nabla\ell(\theta_t) \tag{230}$$

and therefore, **in principle, we can "directly" reach the minimum in 1 step** as follows:

$$\boxed{\theta^* = \theta_0 - H(\theta_0)^{-1}\nabla\ell(\theta_0)} \tag{231}$$

However, we need to 1) be able to calculate the Hessian, and 2) be able to invert it. This is where the problem lies because in neural networks, first, we have a lot of parameters, second, we are not in the quadratic case, so even the previous equation is not exact, and it would require applying the second-order Newton's algorithm. **In practice, it is not possible at all to compute the Hessian, and we can only rely on the first-order algorithm**.

9.2.3 Mini-Batch Normalization

This is a result from the past 4-5 years that is highly effective in the case of neural networks, where we typically have

$$x \longrightarrow F(W.x+b) \tag{232}$$

and we are interested in the Hessian with respect to W. If we denote the Hessian of F with respect to a variable z, $F(z) \to H_z[F](z)$, then the Hessian with respect to W (*note: this is a matrix*) can be written as

$$F(W.x+b) \to x\ H_z[F](Wx+b)\ x^T \tag{233}$$

Let's imagine that $H_z[F](Wx+b)$ is, up to a constant, the identity matrix (*note: recall that we want the ratio of min/max eigenvalues to be close to 1*), then the Hessian with respect to W is given by xx^T, and if we now sum over all the samples, this will involve

$$\sum_i F(W.x_i+b) \to \sum_i x_i\ x_i^T \tag{234}$$

We recognize **the autocorrelation of the data**. Now, this autocorrelation can be represented as an ellipsoid that "encloses" all the data points. **If this ellipsoid has small eigenvalues, then the Hessian is ill-conditioned.** So, we can try to **precondition** by attempting to make the autocorrelation as isotropic as possible. However, we don't want to perform a PCA to find the principal axes of the autocorrelation matrix, especially since this operation would need to be repeated for each layer of the network. Thus, we will work on the "original" axes of the data to compute the mean and ensemble variance:

$$M = \frac{1}{N}\sum_i x_i \qquad S^2 = \frac{1}{N}\sum_i (x_i - M)^2 \tag{235}$$

and for rescaling, we perform the following operation:

$$x_i \to x_i' = \frac{x_i - M}{S} \tag{236}$$

However, in this case, we are changing the variables inside the network. To overcome this issue caused by the conditioning of the Hessian, what is proposed is that at the output of a neuron: we perform rescaling as described above, and we add 2 variables α, β such that $\alpha x_i' + \beta$ to undo this rescaling.

So, after this rescaling, the Hessian is well-conditioned, and the gradient descent works well. **What we don't quite understand is why this works so well!** Before implementing this technique, we used **Drop-out**, which randomly destroys a fraction of the connections between layers.

10. Stochastic Gradient

10.1 Introduction

A quick reminder: in this case, the "average" gradient is computed for one randomly chosen data point as follows:

$$\theta_{t+1} = \theta_t - \alpha \nabla_\theta \ell(\theta, z_{i_t}) \tag{237}$$

and on average,

$$E_{i_t}[\nabla_\theta \ell(\theta, z_{i_t})] = \nabla_\theta \ell(\theta) \tag{238}$$

The evolution of gradient descent over "time" becomes more chaotic, and there is a risk of cycling around the minimum because the error between the optimal direction and

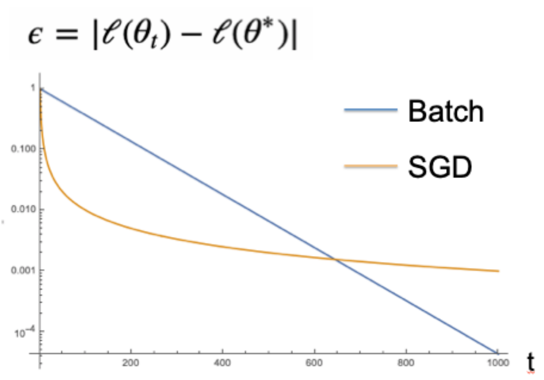


FIGURE 61 – Typical difference between batch gradient descent and a stochastic version: if the stochastic version converges very quickly in the early stages, it becomes much less effective than the batch version beyond a certain number of iterations.

the chaotic direction will increase as we get closer to the minimum.

If we look at the convergence rate, we have the following behaviors[44] (see Figure 61):

$$\epsilon = |\ell(\theta_t) - \ell(\theta^*)| = \begin{cases} O((1-\alpha\mu)^t) & \text{Batch} \\ O(t^{-1}) & \text{SGD} \end{cases} \tag{239}$$

SGD convergence can be faster for the first iterations, but there is a point beyond which Batch is more efficient. However, in the equations above, the step size α is constant. In the next section, let's consider varying it step by step.

10.2 Acceleration of GD and SGD with Variable Steps

If we want to attempt to accelerate the convergence speed of SGD, we need to reduce the "noise", which is the fluctuations of the gradient from one data point to another. One way to do this is to use "mini-batches", which means taking a fraction of the N samples, for example, randomly selecting M variables z_i and averaging the gradient over them. If

44. The evolution in Batch is called "linear" in optimization jargon.

$B = 1$, then we are in the case of SGD, and if $B \to N$, then the convergence will tend towards that of the Batch method.

However, what we would rather have is a method that is like SGD at the beginning and gradually converges towards the Batch method. But there is a more efficient way: varying the gradient step α as we go. Let's see how this works in the case of SGD.

Intuitively, if we go from $t \to t+1$ in 1 step of size α, or if we do the same operation in 2 steps of size $\alpha/2$, then we can compare methods (1) and (2) below:

$$\theta_{t+1} = \theta_t - \alpha \nabla_\theta \ell(\theta_t, z_{i_t}) \qquad (1) \qquad (240)$$

$$\begin{aligned}\theta_{t+1} &= \theta_t - \frac{\alpha}{2}\nabla_\theta \ell(\theta_t, z_{i_{t1}}) - \frac{\alpha}{2}\nabla_\theta \ell(\theta_t, z_{i_{t2}}) \\ &= \theta_t - \alpha\ \frac{1}{2}\left(\nabla_\theta \ell(\theta_t, z_{i_{t1}}) + \nabla_\theta \ell(\theta_t, z_{i_{t2}})\right) \qquad (2) \qquad (241)\end{aligned}$$

By changing $\alpha \to 2 \times \alpha/2$, we have reduced the variance in the gradient estimation by $\sqrt{2}$. Therefore, in the following proof, we will see that in the case of fixed-step SGD, it does not converge, or more precisely, the values will converge within a ball around the minimum, whereas in the case of variable-step SGD, the method converges because the size of the ball tends to 0 as the step also tends to 0.

10.3 Mathematical Framework: Strong Convexity and Regularity

10.3.1 Batch Method

We can demonstrate the "intuitive" result from the previous section in the case of **strong convexity**, even though in practice it also works for neural networks where we have no guarantee of convexity. However, we will place ourselves in a specific framework outlined in three propositions.

— Prop 1: **L-Lipschitz**

We impose a regularity property on $\ell(\theta)$ that is related to the properties of the Hessian as discussed earlier. We impose that **the gradient is L-Lipschitz**, meaning

$$\boxed{\forall \theta, \theta' \qquad ||\nabla_\theta \ell(\theta) - \nabla_\theta(\theta')|| \leq L||\theta - \theta'||} \tag{242}$$

Thus, if ℓ is twice differentiable, then L-Lipschitz is equivalent to $||H(\theta)|| < L$. However, in the case of a ReLU, its derivative is a Heaviside, so the second derivative is a Dirac. Therefore, to accommodate this case, we will not impose double differentiability but stick with L-Lipschitz. Here is a property that follows from the L-Lipschitz constraint:

$$\boxed{\ell(\theta') \leq \ell(\theta) + (\theta' - \theta)^T \nabla \ell(\theta) + \frac{L}{2}||\theta' - \theta||^2} \tag{243}$$

So, with this property, we are actually constraining the change in the *loss* when moving in the gradient direction.

— Prop 2: **Convexity**
Imposing the convexity of ℓ means that

$$\boxed{\forall t \in [0,1] \quad \ell(t\,\theta + (1-t)\theta') \leq t\,\ell(\theta) + (1-t)\ell(\theta')} \tag{244}$$

which means that the value of the *loss* at a point on the line $\{(\theta, \ell(\theta)), (\theta', \ell(\theta'))\}$ is smaller than the weighted average of the values $\ell(\theta)$ and $\ell(\theta')$. If ℓ is twice differentiable, convexity is equivalent to the Hessian being positive, meaning that all of its eigenvalues are positive, and

$$\forall g, \quad g^T H(\theta) g \geq 0$$

This is a property we used in the case of quadratic loss, but here we are dealing with losses that are not twice differentiable, so we need to impose something stronger.

— Prop 3: **Strong Convexity**
We want to avoid cases where the curvature radius at the minimum is very large (i.e., low curvature). This means that we will constrain the smallest eigenvalue of

the Hessian (denoted as μ) to be nonzero. So, consider

$$\boxed{\ell(\theta) - \frac{\mu}{2}||\theta||^2 \quad \text{to be convex.}} \tag{245}$$

In the case where ℓ is twice differentiable, this is equivalent to

$$\mu.Id \leq H(\theta)$$

which we also used in the quadratic loss case.
Strong convexity implies a lower bound property:

$$\boxed{\ell(\theta') \geq \ell(\theta) + (\theta' - \theta)^T \nabla \ell(\theta) + \frac{\mu}{2}||\theta' - \theta||^2} \tag{246}$$

With these three properties, we state the following theorem.

Theorem 8 ***(Batch GD)****:*

If the gradient of ℓ is L-Lipschitz (Prop. 1) and ℓ is μ-strongly convex (Prop. 3), then

$$\boxed{\forall \theta_0,\ \forall \alpha \leq \frac{1}{L} \qquad ||\theta_{t+1} - \theta^*||^2 \leq (1 - \alpha\mu)^{t+1}||\theta_0 - \theta^*||^2} \tag{247}$$

which means that we recover the result from the quadratic case but without imposing that specific form of loss.

Corollary:

We easily obtain a constraint on the convergence to the minimum of the *loss*:

$$\boxed{|\ell(\theta_{t+1}) - \ell(\theta^*)| \leq \frac{L}{2}(1 - \alpha\mu)^{t+1}||\theta_0 - \theta^*||^2} \tag{248}$$

In fact, this corollary can be demonstrated using Prop. 1 and Theorem 8 above. Indeed,

$$\ell(\theta_{t+1}) \leq \ell(\theta^*) + \frac{L}{2}||\theta_{t+1} - \theta^*||^2 \leq \ell(\theta^*) + \frac{L}{2}(1 - \alpha\mu)^{t+1}||\theta_0 - \theta^*||^2 \tag{249}$$

10.3.2 SGD Method

Theorem 9 ***SGD****: In the case of stochastic gradient descent, we start with a regular and strongly convex loss, and we will demonstrate that*

$$\boxed{E(||\theta_t - \theta^*||^2) \leq (1-\alpha\mu)^{t+1}||\theta_0 - \theta^*||^2 + \frac{\alpha}{\mu}B^2} \tag{250}$$

where we make the assumption

$$\forall t \qquad E_{z_i}(||\nabla\ell(\theta_t, z_i)||^2) = \frac{1}{n}\sum_{i=1}^{n}||\nabla\ell(\theta_t, z_i)||^2 \leq B^2 \tag{251}$$

Proof 9. We will proceed with the proof as in the case of Batch:

$$\begin{aligned} ||\theta_{t+1} - \theta^*||^2 &= ||\theta_t - \alpha\nabla\ell(\theta_t, z_{i_t}) - \theta^*||^2 \\ &= ||\theta_t - \theta^*||^2 - 2\alpha\langle\nabla\ell(\theta_t, z_{i_t}), \theta_t - \theta^*\rangle + \alpha^2||\nabla\ell(\theta_t, z_{i_t})||^2 \end{aligned} \tag{252}$$

We want to calculate $E_{i_t}(||\theta_{t+1} - \theta^*||^2)$, which means taking the expectation with respect to the choice of z_{i_t}. We know that

$$E_{i_t}(\nabla\ell(\theta_t, z_{i_t})) = \frac{1}{n}\sum_{i=1}^{n}\nabla\ell(\theta_t, z_i) = \nabla\ell(\theta_t) \tag{253}$$

So, we have

$$E_{i_t}(||\theta_{t+1} - \theta^*||^2) = E(||\theta_t - \theta^*||^2) - 2\alpha\langle\nabla\ell(\theta_t), \theta_t - \theta^*\rangle + \alpha^2 E_{i_t}(||\nabla\ell(\theta_t, z_{i_t})||^2) \tag{254}$$

The term proportional to α is the same as in the Batch case, so we can constrain it in the same way. Thus, we obtain

$$\begin{aligned} E_{i_t}(||\theta_{t+1} - \theta^*||^2) &\leq E(||\theta_t - \theta^*||^2)(1-\alpha\mu) - 2\alpha(\ell(\theta_t) - \ell(\theta^*)) + \alpha^2 E_{i_t}(||\nabla\ell(\theta_t, z_{i_t})||^2) \\ &\leq E(||\theta_t - \theta^*||^2)(1-\alpha\mu) + \alpha^2 E_{i_t}(||\nabla\ell(\theta_t, z_{i_t})||^2) \end{aligned} \tag{255}$$

Now, the quadratic term in α does not necessarily tend toward 0, which is why we introduce the additional assumption that bounds the gradient norm. We then have:

$$E_{i_t}(||\theta_{t+1} - \theta^*||^2) \leq E(||\theta_t - \theta^*||^2)(1 - \alpha\mu) + \alpha^2 B^2 \tag{256}$$

We have a recurrence equation that we can solve exactly:

$$\begin{aligned} E_{i_t}(||\theta_{t+1} - \theta^*||^2) &\leq E(||\theta_0 - \theta^*||^2)(1 - \alpha\mu)^{t+1} + \frac{\alpha B^2}{\mu}(1 - (1 - \alpha\mu)^{t+1}) \\ &\leq E(||\theta_0 - \theta^*||^2)(1 - \alpha\mu)^{t+1} + \frac{\alpha}{\mu} B^2 \end{aligned} \tag{257}$$

This concludes the proof of Theorem 9. ■

Therefore, the theorem tells us that **the gradient term cannot be controlled and will, except for accidents, prevent convergence**. When $t \to \infty$, the term $(1-\alpha\mu)^{t+1}$ tends to 0, but we will tend towards an asymptote for the error in the average of $||\theta_{t+1} - \theta^*||^2$. To overcome this, we need to make α decrease towards 0. How should we choose it?

— Prop 5. We will choose $\alpha = \alpha(t)$ such that

$$\boxed{\alpha(t) = \frac{\mu}{1 + \mu^2 t} \Rightarrow E_{i_t}(||\theta_{t+1} - \theta^*||^2) \leq \frac{C}{1 + t\mu} = O\left(\frac{1}{t}\right)} \tag{258}$$

Indeed, if in the equation of Theorem 9, we identify α as $\alpha(t+1)$ and we want

$$(1 - \alpha(t+1)\mu)^{t+1} \sim \frac{\alpha(t+1)}{\mu} \tag{259}$$

then if $\alpha(t)\mu \ll 1$ (not fundamentally necessary), we have

$$\alpha(t) = \frac{\mu}{1 + \mu^2 t} \tag{260}$$

So,

$$E_{i_t}(||\theta_{t+1} - \theta^*||^2) \leq E(||\theta_0 - \theta^*||^2)(1 - \alpha\mu)^{t+1} + \frac{\alpha B^2}{\mu} = (E(||\theta_0 - \theta^*||^2) + B)\frac{\alpha}{\mu} \quad (261)$$

$$\leq \frac{C}{1 + \mu^2 t} = O(1/t) \quad (262)$$

This gives us the convergence rate of SGD.

10.4 Generalizations?

Can we generalize the ideas from the previous sections?

— Non-differentiable framework

If we have non-differentiable points (for $\ell(\theta)$), it's as if the gradient is not unique (notion of "sub-gradient"). But these gradients "remain" below the curve, which is convex, so this doesn't change the previous proof as long as we constrain that the average of the "sub-gradients" is smaller than B^2.

— Number of operations p. In the case of SGD, the error on θ is typically $\epsilon \sim O(1/p)$, while in Batch, $\epsilon \sim O(1 - \alpha\mu)^{p/n}$.

Referring to Figure 61, we can see that there is an "optimal" point. If n is large and we don't need extreme precision, then the stochastic method is your friend; if n is not too large, it's better to use the Batch method.

There are hybrid methods that start with SGD and switch to a Batch-like scaling after a certain point (see Francis Bach's work, including the SAG and SAGA methods).

But in the case of neural networks, these methods don't address the core of the problem. Why? **Mainly due to local minima, and we are not in a strongly convex framework**. However, what we observe experimentally is that we are not blocked by the asymptotic speed of SGD, but rather by local minima. It's a much more challenging problem to control optimization, especially when deciding on the network's architecture.

10.5 Optimization Problems

Let's list a number of points that remain unresolved (non-exhaustive):

1) The loss function $\ell(\theta)$ is not convex, so it's necessary to understand the structure of local minima. For example, what is their profile? Are they narrow or wide? This underlies the sensitivity to a step in θ. We can try to determine their number, find out if they are deep, etc. In fact, we encounter problems quite similar to those in **Statistical Physics**, where when we lower the temperature of a system, we want to ensure that it falls into a stable state of minimal energy.
2) Learning rates (α): we know how to control them in the strongly convex case, but in other cases, it's often an experimental trial-and-error process.
3) Mini-batches work, but we don't understand why when we try to comprehend how this cascades in deep networks.
4) It shouldn't be believed that convergence is achieved every time, far from it, and there are many counterexamples. For example, in the article arXiv:1812.01662, it is shown that the neural network does not converge at all for a simple task of binary pattern comparison that could *a priori* be considered a simple task. But there are misconceptions, for example, "in very high dimensions, there is little chance of falling into a local minimum" (FALSE).

The strategies to address these problems fall into two categories:

— Either we want (try) to "control everything", including the number and structure of local minima, etc.

— Or we proceed in the manner of the study of "dynamical systems" that focus on the trajectory in "real-time" minimization so that the probability of falling into a local minimum with $\ell(\theta)$ far from $\ell(\theta^*)$ is low.

These reflections conclude this course for the year 2019, and we still have many unanswered questions. Next year, we will delve into convolutional networks, which seem to be suitable for capturing very global regularities in high dimensions in all the domains where they have been introduced.