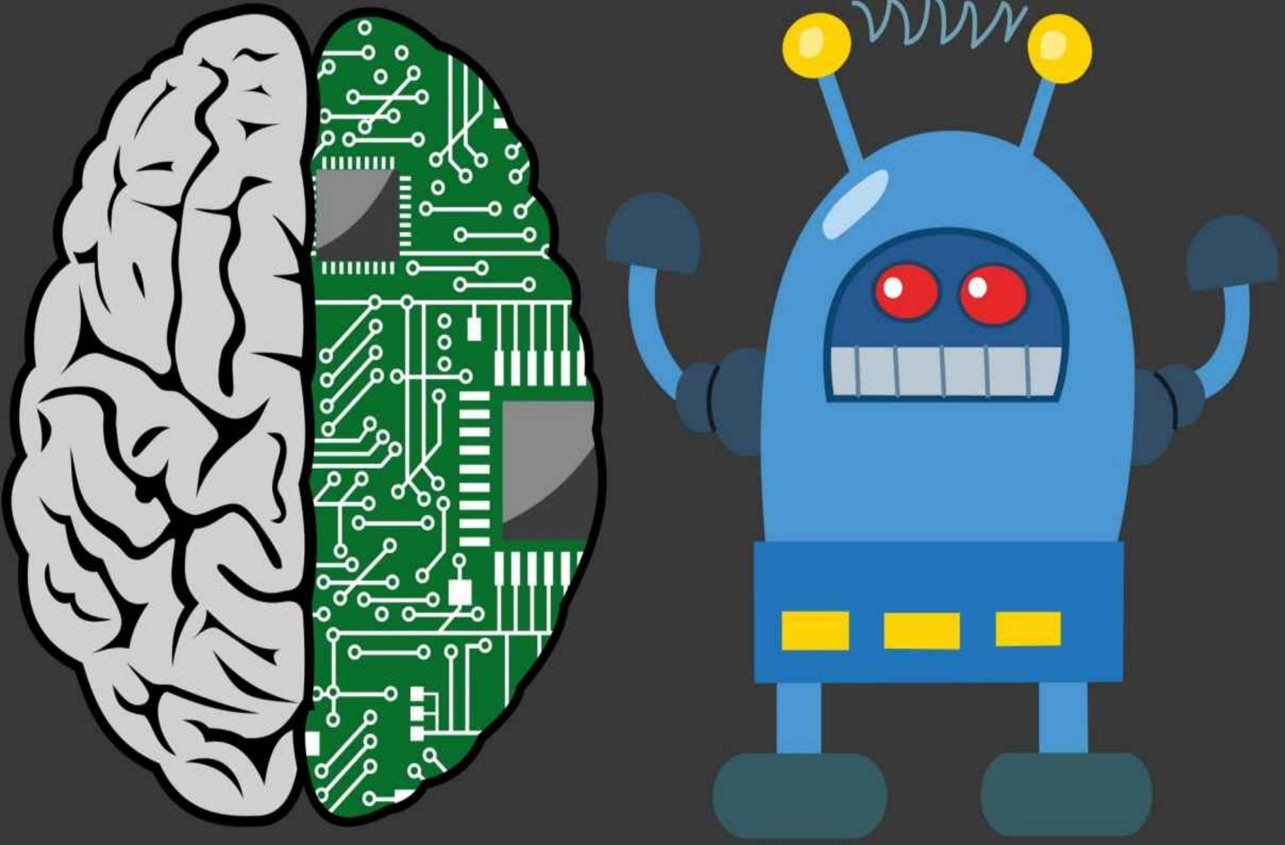
NEURAL NETWORKS

[AND (DEEP) + (L)EARNING]

EXPLAINED TO YOUR GRANNY



Pat Nakamoto

Neural Networks & Deep Learning **Deep Learning explained to your granny – A visual introduction for beginners who want to make their own Deep Learning Neural Network**

By

Pat Nakamoto

Contents

Introduction

“A sinister threat is brewing deep inside the technology laboratories of Silicon Valley. Artificial Intelligence, disguised as helpful digital assistants and self-driving vehicles, is gaining a foothold -- and it could one day spell the end for mankind”.

ELLIE ZOLFAGHARIFARD, Mail Online

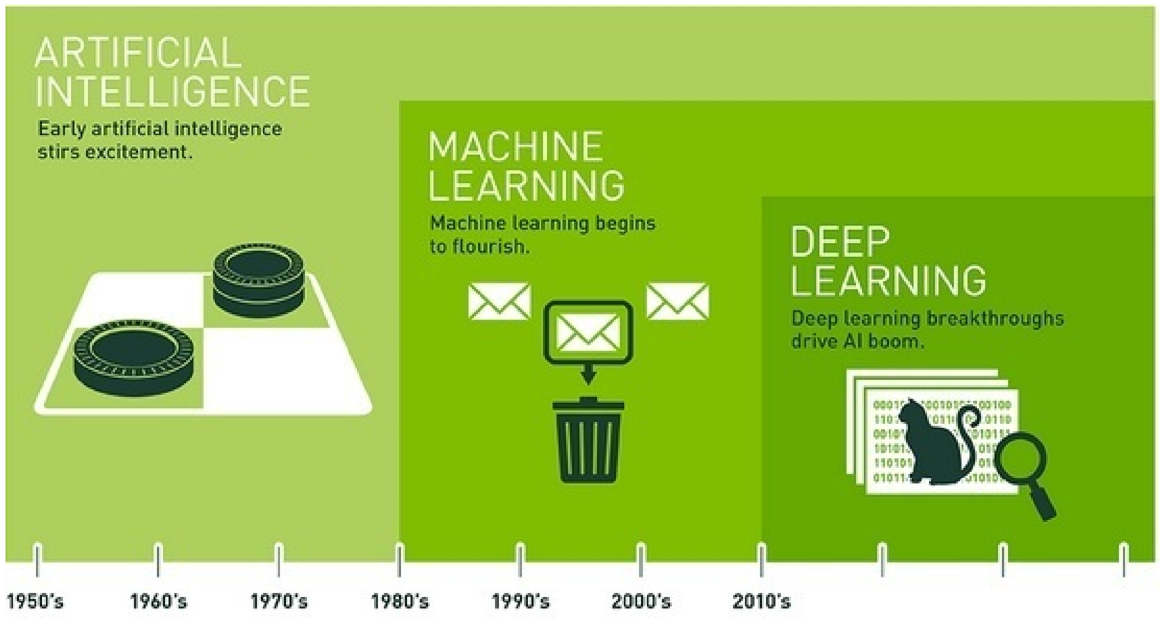
You certainly have noticed that Machine Learning and Artificial Intelligence are seemingly the biggest technological hype waves of the moment. World giant players, such as Apple, Amazon, IBM, Google and Facebook, heavily invest in machine learning research and applications for good reasons. However, what is this hype all about?

Machine Learning applications are already very numerous today, and some have entered our daily lives without us actually realizing it. For example, think of search engines: when you insert one or more keyword, a search engine returns a list of result, called SERP (Search Engine Results Pages) that are the effect of Machine Learning algorithms using unsupervised learning (providing as output information deemed relevant for the research carried out, based on the analysis of schemas, models, and structures in data).

Another common example is email spam filters based on Machine Learning systems that learn continuously to either intercept suspicious or fraudulent e-mail messages and act accordingly (such as deleting them before they are deployed on the user’s personal mailbox). Systems of this type, even with greater sophistication, are used in the financial sector for the prevention of fraud (such as credit card cloning) and the theft of identity data.

Algorithms learn to act by correlating events, user habits, spending preferences, etc. Through this information, algorithms can identify in real-time any abnormal behavior that could actually identify a theft or fraud. We can also find interesting examples in the medical field, where algorithms learn to make accurate predictions to prevent epidemic outbreaks or to diagnose cancer or rare diseases in an accurate and timely manner.

Neural Networks and Deep Learning


ARTIFICIAL INTELLIGENCE
Early artificial intelligence stirs excitement.
MACHINE LEARNING
Machine learning begins to flourish.
DEEP LEARNING
Deep learning breakthroughs drive AI boom.
1950's
1960's
1970's
1980's
1990's
2000's
2010's

Since their birth in the ‘50ies, **Artificial Neural Networks** have been labeled by experts as one of the most promising areas of science within the Machine Learning field. (Artificial) Neural Networks are computational models inspired by the functioning of the human brain. Their main feature is the ability to learn during a training phase and then generalize the knowledge acquired to predict new situations. Just like a human brain, these networks have an internal memory that increases through experience.

Deep learning is a term that indicates a particular approach to the design, development, testing and especially training of neural networks. The birth and development of Deep Learning has been made possible thanks to the technological improvements that have appeared at the end of the 2000s, such as GPU, that is, large arrays of small processors designed to process images in video cards of Nvidia. This technology, initially created for video games, was then discovered to be very fast in the processing of multilayer neural networks, with much better performances than normal CPUs. The second innovation that has determined the definitive advent of deep learning is the Big Data phenomenon, that is the availability of massive and kaleidoscopic amounts of data thanks to the diffusion and growth of the Internet and of the services connected to it. In fact, now there are billions of images ready, as test data, both marked and unmarked, which can be used for developing experience programs; years of telephone exchanges registered, to have audio samples of thousands of different voices, millions of digitized texts available…in a few words a ‘planet database’ containing Zettabytes of information to use!

Before diving into this exciting world, I want to explain that the goal of this book is to offer information that is highly informative, yet accessible to anyone - which entails using many generalizations, so bear with me, generalizations/some degree of simplification may be necessary. Nevertheless, if this guide can make someone more passionate about Deep Learning, my mission can be considered as fulfilled.

Chapter 1. A brief introduction to Machine Learning

Notes to this Chapter

The aim of this Chapter is to introduce the general concepts of machine learning, the two main types of learning and some basic terminology, as general basis for introducing Neural Networks in Chapter 2.

What is Machine Learning?

Imagine being a professor and entering a class in which students are ll computer machines, all sitting neatly with an apron and lunch-box, ready for you to teach. Your role, in that classroom, is to teach them to make decisions autonomously. B eing a regular teacher will not help you to succeed. In order to make your computer-students autonomous, you will need to know algorithms in the field of **automatic learning**. This field has often been associated with artificial intelligence, and more specifically, **computational intelligence**. Computational intelligence is a method of data analysis that points to the automatic creation of analytic models. That is, allowing a computer to work out concepts, evaluate decisions, take decisions, and predict future options. Obviously, these choices must be able to adapt to new situations and not be standard. Systems based on this type of learning are at the basis of the development of self-driving cars, for example. Machine Learning hence has to do with the ability of a computer to learn, given a input array of data (for example data collected through sensors, GPS, etc.) and through algorithms, how to recognize the surrounding environment adapting their "behavior" according to the specific situations they face (becoming able even to drive a car).

Yet, for a novice, the theoretical concepts behind machine learning can be quite overwhelming. Of course, we cannot discuss all the nitty-gritty details about all the different algorithms and applications that have emerged in the last 65 years, however, in this book, we will embark on an exciting journey that covers some relevant basics.

So, Machine Learning refers to the fact that there are generic algorithms that can give interesting information about a certain data set without you having to write any specific code for the problem. Instead of writing code, the data is inserted into a generic algorithm and the algorithm generates its own logic based on the input. For example, one of these algorithms is the Classification Algorithm. I can enter mixed heterogeneous data, and my algorithm is able to identify autonomously common features classifying the data into several groups. This same classification algorithm can be used for purposes that range from recognizing handwritten numbers to classifying e-mails, indicating whether they are spam or not, without having to change any line of code. It is the same algorithm, but it is powered by different Training Data so that different classification logic emerges. Machine Learning is an umbrella term that includes many of these generic algorithms.

Two main Types of Machine Learning Algorithms

Machine Learning algorithms can be divided in two main categories - Supervised Learning and Unsupervised Learning algorithms. The difference is simple but very important:

- **Supervised Learning algorithms**: Let us suppose you are the owner of a car dealership centre. Your business is growing, and you have to hire new staff and train them to help you. But there is a problem – it just takes a glance to you at a certain used car to havea clear idea of how much it is worth, but your new employees do not have a clue, since they do not have your experience in estimating car prices. To help your employees (and perhaps get a few free days to go on vacation), you decide to write a small application that can estimate the value of a car based on its registration year, engine size, travelled kilometers and the average selling price of the same brand of car. For 3 months, you register the price at which certain models of cars sell, writing down the details of the transaction – year of the car, model, kilometers, power , etc., and last but not least the final sale price. Using this training data, you can create a program that can estimate what could be the right sales price of any other car of the same models (let's say Mercedes and Audi). This is called supervised learning. You know the price at which each car has been sold, so knowing the previous answers to the problem, you are able to work backwards and understand the logic to apply in order to solve new similar problems. To build your application, you have to feed the Machine Learning algorithm with the data you collected regarding each car. The algorithm will try to understand what kind of mathematical functions it must use to produce the solution for new problems. Once you know what math function applies to a specific set of problems, you will be able to produce a solution for any other problem of the same type!
- **Unsupervised Learning:** Let us go back to our example with the car-dealership center owner. How would you create the same application without

knowing the sales price of each new car? Even if you know only the kilometers and model of each car, you can yield interesting results….this is unsupervised learning. It's like someone giving you a list of numbers on a piece of paper and telling you, "I do not really know what these numbers mean, but maybe you can figure out if there's a scheme or code or something at the base of them - Have fun! ".

So, what could you do with this data? To begin with, you might have an algorithm that automatically identifies the different market segments in the data. You might find that Audis are bought at higher prices below a certain mileage, but Mercedes's can be priced like gold even over a certain number of kilometers. Knowing these different types of customer choices, can help you better manage your marketing work.

Another interesting thing you could do is to automatically identify any abnormal values that are very different from all the others. Maybe Audis that are sold at abnormally high prices all are of a certain model and year, and you can try to concentrate your salespersons on promoting those so you get higher commissions.

Other than these, there are also other kinds of Machine Learning algorithms:

- **Reinforcement Learning algorithms**: In this case, the system has to interact with a dynamic environment (which allows it to have input data) and reach a goal (receiving a reward accordingly), also learning from errors (identified through "punishments"). The behavior (and performance) of the system is determined by a routine of learning based on reward and punishment. With such a model, the computer learns, for example, to beat a rival in a game (or to drive a vehicle) by concentrating efforts on carrying out a given task. While doing this, it is aiming at reaching the maximum value of the reward; in other words, the system learns to play (or to drive) by improving performance as a result of previously achieved results.
- **Semi-supervised learning**: this is a "hybrid" model in which the computer is

provided with an incomplete set of training / learning data; some of these inputs are "endowed with" their respective output examples (as in supervised learning), while others are missing (as in unsupervised learning). The underlying objective is always the same: identify problem-solving rules and functions as well as data structures that are useful for achieving certain goals.

- **Other Practical Approaches in Machine Learning**: There are also other Machine Learning subcategories, if we are thinking in "practical" terms. These approaches range from probabilistic models to Deep Learning, which is the main topic of this book. For example, we can think of the so-called "**tree of decisions**", based on graphs through which you can develop predictive models that enable you to discover the output of certain input decisions. Another concrete example is "**clustering**", or mathematical models that allow you to group data, information, objects, and so on according to their “similarity". There is then the sub-category of "**probabilistic models**", which base the learning process on the calculation of probabilities. The most known is the "**Bayes network**", a probabilistic model that represents in a graph a set of random variables and its conditional dependencies (a relationship between two or more events that are dependent when a third event occurs). Finally, there are artificial **neural networks** that use certain algorithms to learn inspired by the structure, functioning and connections of biological neural networks (i.e. those in the human being). In the case of so-called multi-layer neural networks, you enter the field of **Deep Learning**. The latter two are the topics of this book.

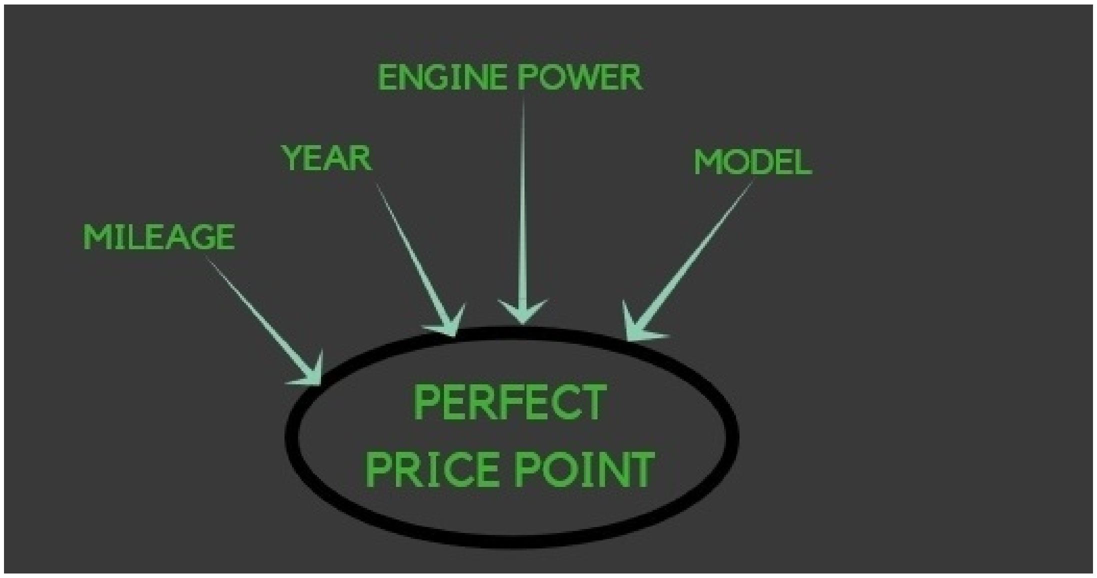
A practical example of Unsupervised Learning

“Great, but being able to estimate the price of a car can be really considered ‘learning’?” you might be thinking. In humans, the brain can approach any situation and learn how to deal with it without explicit instructions. If you sell groceries for a long time, you will surely get the "feeling" about the right price for a certain car, the best way to market it, the kind of customer that might be interested in it, etc. The goal of Strong AI (Strong Artificial Intelligence) research is to be able to replicate this ability on computers.

To date, Machine Learning algorithms are not so advanced - they only work when they face a very specific, limited problem. In this case, perhaps a more appropriate definition of "learning" would be “understanding an equation to solve a specific problem, based on a sample of data." Unfortunately, "Automation that allows you to understand an equation that solves a specific problem, based on a sample of data" is not a name your granny would understand. Therefore, we have to call it ‘**Machine Learning’**.

So, how do we write the program to estimate the value of a car as in our example? Think it out for a moment before you go ahead reading.

If you do not know anything about Machine Learning, you will probably try to write some basic rules to estimate the price of a car; you can spend hours and hours over it, and eventually get something done. However, the program will never be perfect and it will be difficult to keep it updated as prices change constantly. Would it not be better if your computer could figure out how to solve this issue in your place? We do not care what the function exactly does, but that it produces the correct values. You can think of this as if the price was a delicious stew and the ingredients were the mileage, the year, the size of the engine and the model of the car. If you understand how much each ingredient impacts on the final price, there is probably an exact ratio among the ingredients that produces the perfect price.



This would reduce the original function (with all its 'if' and 'otherwise') to something much simpler: by identifying the elements that weigh on our final result, we can adjust the weight of each element until we find the perfect combination, and our function will be able to automatically predict car prices!

A simple way to understand the best ratio of your elements is: Step 1: Start with the weight of every element set to 1.0.

Step 2: Use the function for each car you have pre-existing data for and see how your function’s results vary from the correct real price. For example, if the first car was actually sold at $ 25.000 dollars, and the function produced as a result $ 15.000, there is a difference of $ 10.000 for that particular car.

Step 3: Now sum up the square value of the difference related to each car result and real selling price you have in your data set. Let us say you have 500 car sales in your data set, and the total difference your function has processed is $ 8.000. This value indicates how "wrong" your function is at present.

Step 4: Now, take that total sum and divide it by 500, to get an average difference for each car. Call this average amount your ‘wrong function cost’.

If you could take this cost down to zero, playing with the weight ratio of the different elements in your function, the function would be perfect. It would mean that in all cases, the function perfectly guesses the car price based on the input data.

Step 5: Repeat the second step over and over again with every possible combination of weights. Whatever combination of weights makes the cost closer to zero, is the one to use. When you find the weight that works, you have solved the problem!

Simple, right? Well, think about what you just did. You had some data, you fed the data into three large sets, you carried out some simple steps, and it all ended up with a function that can guess the most convenient price of a car. Wow! However, here are some facts that will leave you even more astonished:

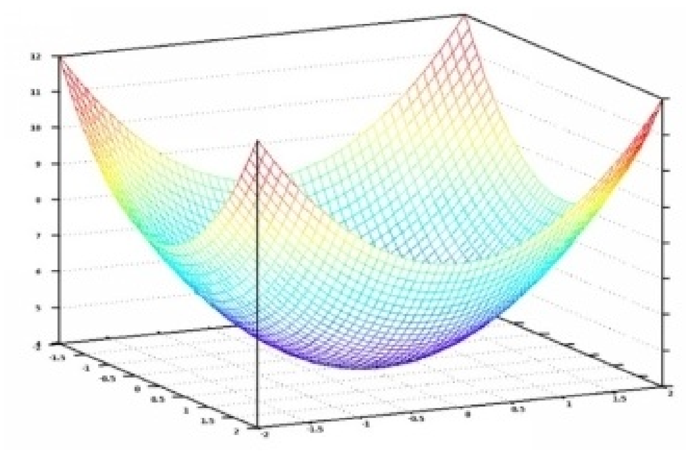
- Research in many fields (such as linguistics / translation) over the last 40 years has shown that these generic learning algorithms that "mix data" have better results than those generated by real people who use explicit rules. The "silly" approach of Machine Learning ultimately beats human experts (the algorithm does not know ‘why’ it is attributing more weight to an element rather than to another, as a senior experienced store owner would know). Have you ever used Google Translator? I bet you have, and it will allow you to understand exactly what I mean. The program takes the information it needs for processing the translations from unfiltered web pages, full of spelling and syntactic errors, and which sometimes are even incomplete. Yet the overwhelming amount of data available allows the program to be more reliable than all its predecessors, which were based on correct dictionaries drafted by experts, but containing a limited amount of information. The function we spoke about in this paragraph is actually ‘silly’. The algorithm does not know what the supplier’s origin or freshness are; all it knows is that it needs to mix these values to output the correct answer.
- It is very likely that you have no idea why a particular combination of weights works. Therefore, you wrote a function that you do not fully

understand, but nevertheless works perfectly.

- Imagine that instead of taking the period of the year and the supplier’s origin as parameters, the function includes a range of numbers. Let us say that each number represents the brightness of a pixel of an image captured by the camera mounted on the top of your car. Now let us say that instead of looking for a value called "price", the function outputs a forecast called "steeringwheel_rotation_degrees". You just have created a function that can drive a car on its own! Incredible, right? Your granny would be astonished.

But what about the "try every possible ratio" thing we spoke about in Step 3? Okay, of course you cannot try all the existing ratio combinations of all the possible weight features to find the best combination. If you were to take this literally, you would continue *ad infinitum* because you would never fall short of numbers to try. To avoid this, mathematics has figured out many clever ways to quickly find good values for those weight features without having to do many attempts. For example, try this. First, write a simple equation that represents Step # 2:

$$\text{Cost} = \sum_{i=1}^{100} ((\text{MyGuess}(i) - \text{RealAnswer}(i))$$

This equation represents *how wrong* our estimate is of weight of the various elements that impact the final price. If we were to trace this cost equation for all the possible values of our weight features, we would get a graph that might look like the one below: 

We just need to adjust our weights to make the equation move towards the middle, the lowest part of the chart. If we continue to make small adjustments on our weights by moving them towards the lowest point, we will reach our goal without having to try a great number of other ratios. If you remember something about calculus, you might remember that the derivative of a function indicates the inclination of the tangent to the function at any point. In other words, it tells us that the direction is down for any point on our chart. We can use this concept to continue the walk downhill. Therefore, if we calculate a partial derivative of our cost function relatedto each of our weights, we can subtract that value from each weight. This will allow us to walk closer and closer to the bottom of the chart. Continue to do so and eventually you will reach the bottom where you have the best possible values for our weights. This is a summary of a way to find the best weights for your function called **Batch Gradient Descent**.

The great thing is that when you use Machine Learning to solve a real problem, all these calculations will be done for you by your computer. Nevertheless, it isalways useful to have a good idea of what is going on!

I did skip some details in this explanation, for simplicity and convenience. The three-phase algorithm I described above is called **Multivariate Linear Regression**. You are estimating the equation of a line that fits all of the points of the data related to your car. When you use this equation to estimate car sales price, you do not know where it will appear on the line. This is really a powerful idea with which you can solve "real" problems.

The approach I have shown can work in the simplest cases, but not in *all* cases. One reason is that produce prices are not always fairly simple and straightforward so to follow a continuous straight line. However, luckily there are many ways to handle this problem. Many other Machine Learning algorithms, as we will see, can handle nonlinear data (such as neural networks or Support Vector Machines with kernels). There are also ways to use linear regression in a smarter way that allows you to handle the most complicated lines. In all cases, the basic idea of having to find the best combinations always applies.

In addition, I ignored the idea of **overfitting**. Overfitting refers to the risk of “excessive adaptation” of your network, which usually occurs in machine learning or statistics when a very complex statistical model adapts to the observed data (the sample) because it has too many parameters with respect to the number of observations.

It is easy to come up with the ratio of a series of weights that always works perfectly to predict car prices in the original data set, but it does not work for all the new cars that were not present in the original data set. However, there are ways to deal with this problem too (such as regularization/validation and data set cross-validation). Learning how to deal with this problem is a key element to learn how to apply Machine Learning successfully. In other words, while the basic concept is quite simple, it takes some skill and experience to apply Machine Learning and to get truly useful results. But it is a skill that any developer can learn!

Once you begin to understand the simplicity with which Machine Learning techniques can be applied to problems that seem really complex (such as handwriting recognition), you begin feeling as if you could use Machine Learning to solve any existing problem and get a solution as long as you have enough data. Just enter data and look at the computer to calculate magically the equation that dividesthem! However, it is important to stress again that Machine Learning works only if the problem is solvable with the data you have available. For example, a model that predicts car prices based on the color of the car, will never work. There is no relationship between the color of a car and its selling price. Therefore, no matter how you feel, your computer cannot infer a relationship between the two!

So remember, if an expert human cannot use the data to solve the problem manually, even a computer will probably not be able to solve it. Instead, focus on problems a human being *can* solve, but it would be better if a computer could solve them, for instance because it would be much faster.

In my opinion, at this time the biggest problem with Machine Learning is that it lives mostly in the academic world and research groups. For non-experts who wish to have a wider view on the subject, it is not easy to understand the material about it. Nevertheless, every day this is getting better. The free work of Andrew Ng Machine Learning Class on *Coursera* is amazing. I would highly recommend going there. It should be understandable for anyone who has a degree in computer science and remembers a minimum of math. In addition, you can play with tons of Machine Learning algorithms, downloading and installing SciKit-Learn. It is a python framework that has a "black box" versions of all standard algorithms.

Key points of this Chapter

- Machine Learning refers to the fact that you can feed data into a generic algorithm and the algorithm generates its own logic based on the input without you having to write any specific code for the problem;
- There are two main categories in Machine Learning algorithms - Supervised Learning and Unsupervised Learning;
- We have Unsupervised learning when you only have input data (X) and no corresponding output variables. We have Supervised learning when you have input variables (x) and an output variable (Y) and you use an algorithm to learn the mapping function from the input to the output.

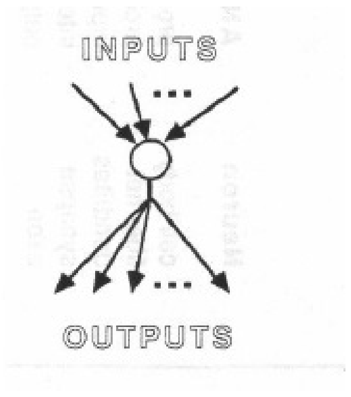
Chapter 2. Neural Networks

Notes to this Chapter

In the previous paragraph, we have said that <u>Machine Learning uses generic algorithms to extrapolate interesting information about data without having to write specific code for the problem that you want to solve</u>. Now, we will work on building a very specific implementation: a Neural Network!

What are Neural Networks?

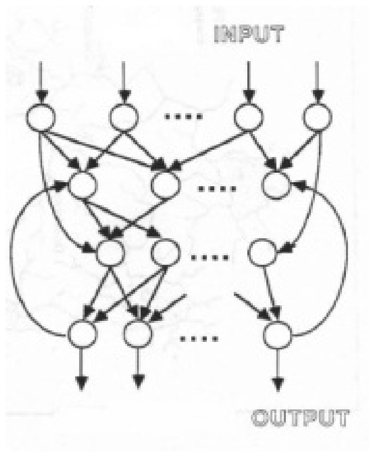
So, what are Artificial Neural Networks? They are a new computer technology: think of many simple parallel processors, strongly integrated by a network of connections that create a computational distributed model.



Single processing unit

Their architecture hence was quite innovative in the ‘50ies, when they were born in clear analogy with the structure of the brain: many neurons (1010) strongly connected by synapses through which the computations spread in parallel in the cerebral cortex. This allows achieving new performances: finding the solution in real time to complex problems, self-learning, resistance to faults and errors, etc.

Neural Network



But how do Neural Networks work exactly?

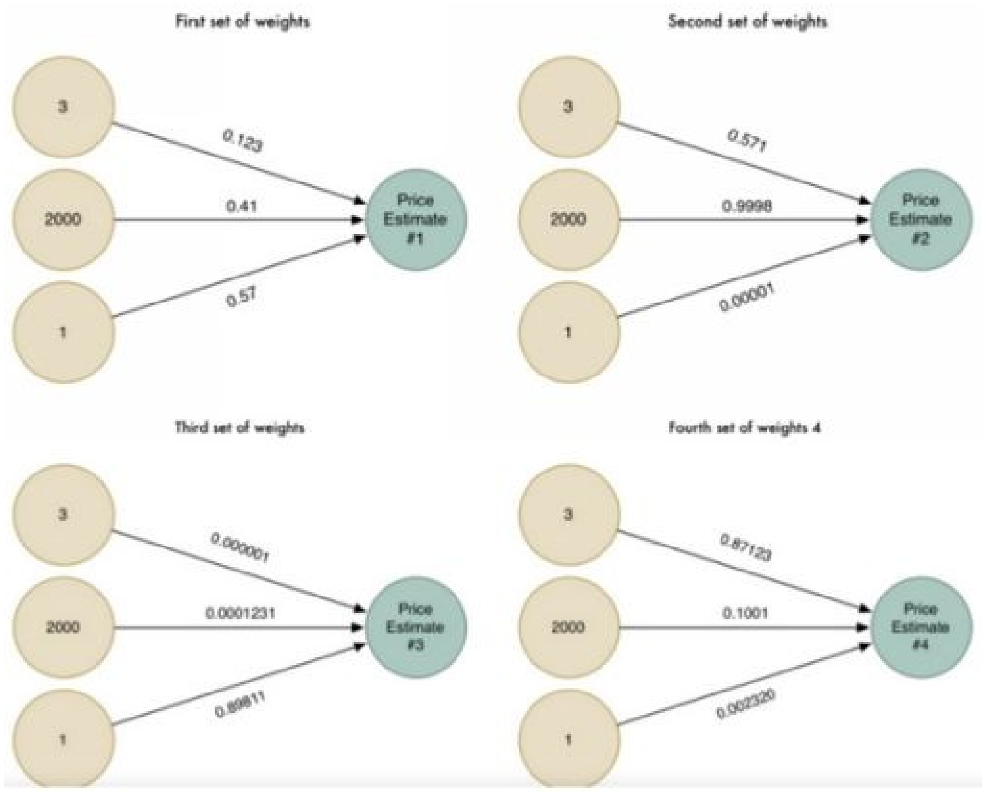
A Neural Network consists of numerous homogeneous processing units, strongly interconnected through links of varying intensity. The activity of the single unit is simple, and the power of the model lies in the configuration of connections (topology and intensity). Starting from the input units, to which the data is fed in order to solve a problem, the computation propagates in parallel within the network up to the output units, which provide the result. A Neural Network is not programmed to execute a certain single activity, but trained (using an automatic learning algorithm) by means of a series of examples of the reality to model.

Back in the ‘50ies, Neural Networks had a simple structure with few internal units, while the current ones involve millions of units, being able to learn incredibly complex patterns, even if they require much more powerful computers and more sophisticated training techniques.

If we were to describe the functioning of an Artificial Neural Network in a sentence, we could say that it takes in input data, and ‘makes sense of it’, finding regularities, or patterns.

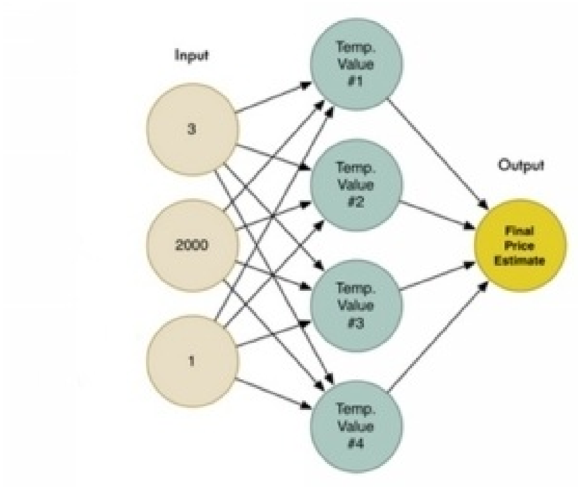
As you are reading this book, your brain is organizing letters into words, and words into sentences, until the meaning emerges. Likewise, an Artificial Neural Network can analyze your comments on a website in search of meaningful words and relevant topics. This drive to make sense of information is so deeply inherent in our brain, that it is also applied to meaningless patterns. When we look at the clouds, we spontaneously associate them with real images and categories. This same mechanism is present in Artificial Neural Networks, which make them extremely fascinating and at the same time frightening.

In the previous paragraph, we created a simple algorithm that calculates the value of a car based on some of its features. In other words, we have weighed the value of the car by multiplying each of its features by a certain weight value. Then we summed up these numbers until we obtained the value of the car. Instead of using code, we tried to represent this function using a simple diagram; however, this algorithm only works for very simple problems, for which the result has a linear relationship with the input data. What if there was not a simple relationship behind the determination of car prices? For example, maybe the supplier’s origin has a lot of relevance for the price of local produce, but it does not matter at all in the case of exotic produce. How could we handle this kind of peculiarity in our model? To get a better result, we could run this algorithm several times with different weights to catalog different limit-cases.



So now, we have four different estimates. Let us combine these four results to get a single final price. Let us put them all in the same algorithm (but using another combination of weights)! Our new super-solution combines the estimates of our four different attempts to solve the problem. Thanks to this, we can model more cases than we could capture with a simple pattern.

Let us unite our four attempts in one big scheme: this is a neural network!



Each knotknows how to take a set of values as input, apply weights to them, and calculate an output value. By linking together many knots like these, we can model very complex functions.

I am making many generalizations to keep the explanation simple (for example, I am not talking about the **scaling feature** - and about the **activation function**) but the most important part is based on these ideas:

- We have created a simple estimation function that takes a number of inputs and multiplies them for various weights to get an output. We call this simple function a **neuron**.
- By linking many simple neurons together, we can model functions that would be too complicated to be managed by a single neuron.

It is just like LEGO! We cannot model as much with a single LEGO cube, but we can model anything if we have enough cubes of LEGO to unite.

McCulloch-Pitts's Neuron

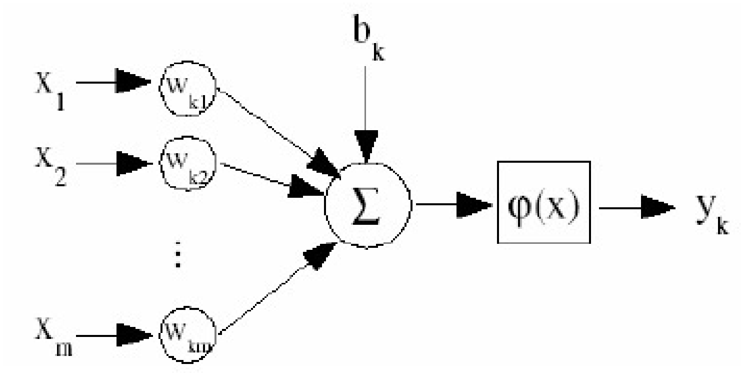
So, up to now we have said that Artificial Neural Networks are computational systems inspired by the biological processes that occur in the human brain. Many of their features in fact are inspired by biological processes: • they are formed by millions of computational units (called neurons) able to execute a pondered sum; • they have a high number of weighed connections (synapses) among the units; • they are highly parallel and non-linear; • •they are adaptive and trainable and learning is done by changing the connections' weights; • they are error-tolerant because storage is widespread; • there is no distinction between memory and calculation area; • they have generalization skills: they can produce reasonable outputs with inputs that they never have encountered before during the learning process.

Wow!...exciting. Let us dive into the details of how a single unit works, the McCulloch-Pitts's Neuron.

A neuron is the fundamental unit of calculation of a Neural Network, and is made up of 3 basic elements in this model:

1. a set of synapses or connections each of which is characterized by a weight (synaptic efficacy); unlike the human model, the artificial model can have both negative and positive weights;
2. a summing agent that sums up the input signals weighed by the respective synapses, producing a linear combination of the inputs;
3. an activation function to limit the breadth of a neuron's output. Typically for convenience the width of the outputs belongs to the interval [0,1] or [-1,1].

The neuronal model also includes a threshold value that has the effect, depending on its positivity or negativity, of increasing or decreasing the net input to the activation function.



In mathematical terms we describe a k neuron with the following equations:

$$u_k = \sum_{j=1}^{m} w_{kj} \cdot x_j$$

$$y_k = \varphi(u_k + b_k)$$

Where:

- x_i are the synaptic weights of neuron k; • u_k is the linear combination of the inputs in the neuron k; • b_k is the threshold value of the neuron k; • φ (x) is the activation function; • y_k is the output generated by the neuron k.

We can reformulate the 2 equations incorporating the threshold value in uk thus

$$v_k = \sum_{j=0}^{m} w_{kj} \cdot x_j$$

obtaining a model equivalent to the previous one:

$$y_k = \varphi(v_k)$$

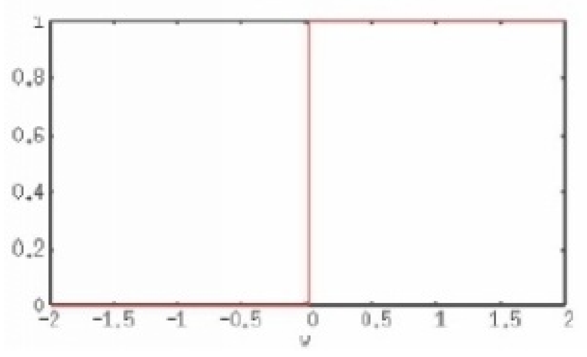
Where the input x_0 = 1 and wk_0 = bk. We define vk activation potential.

Types of activation function

We identify 3 types of basic activation functions: Threshold functions or Heaviside functions, Piecewise-linear functions, and Sigmoid functions.

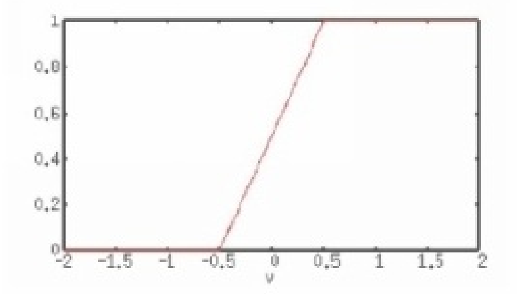
Threshold functionsare used in the McCulloch-Pitts neuron model.

$$\varphi(v) = \begin{cases} 1 & \text{se } v \geq 0 \\ 0 & \text{se } v < 0 \end{cases}$$



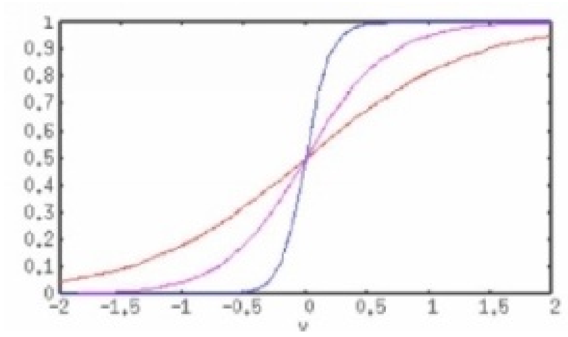
$$\varphi(v) = \begin{cases} 1 & \text{se } v \geq \frac{1}{2} \\ v & \text{se -}\frac{1}{2} < v < \frac{1}{2} \\ 0 & \text{se } v \leq -\frac{1}{2} \end{cases}$$

Piecewise-linear functions are as following:



Sigmoid functions are the most used function in the creation of artificial neural networks. It is a strictly growing function that exhibits a balance between linear and non-linear behavior. For example:

$$\varphi(v) = \frac{1}{1 + e^{-a \cdot v}}$$

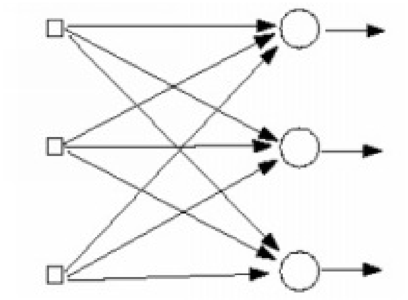


where *a* is a parameter that indicates the slope of the function.

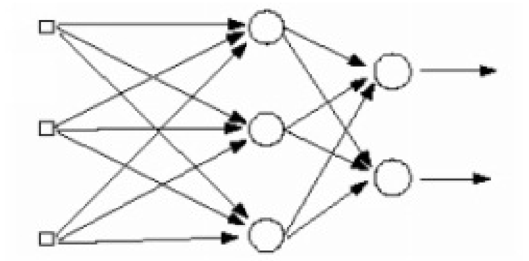
Types of network architectures

The way a network is structured depends on the learning algorithm that you intend to use. In general we can identify 3 classes of networks.

1. **One layer Feedforward networks.** In this simple form of layered network, we have input knots and a layer of neurons (output layer). The signal propagates through the network in an linear way, starting from the input layer and ending in the output one. There are no connections that come back and no transversal connections in the output layer.



2. **Multilayer feedforward networks.** This class of feedforward networks differs from the previous one since it has one or more layers of hidden neurons (hidden layers) between the input and output layers. Each layer has incoming connections from the previous layer and exiting into the following one, therefore the signal propagation takes place linearly without cycles and without transversal connections. This type of architecture provides the network with a global perspective as it increases interactions between neurons.



3. **Recurring networks** (feedback). A recurring network differs from previous ones in the fact that it is cyclical. The presence of cycles has a profound impact on the learning abilities of the network and on its

performances, in particular make the system dynamic.

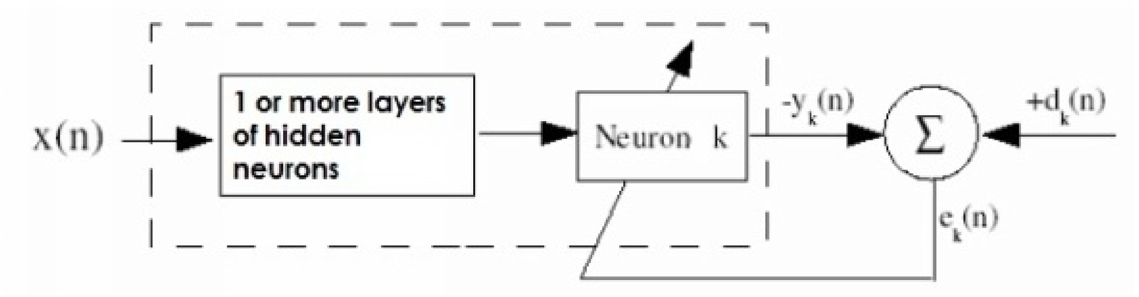
Learning processes

Learning is the process through which the free parameters of a neural network adapt, using a stimulation process, to their surrounding environment. The type of learning is determined by the way these adaptations take place.

A **learning algorithm** is a set of well-defined rules that solve a certain learning problem. As we expalined in Chapter 1, the main two types of learning algorithm are: • Supervised: thre is a =’teacher’ who knows the environment and provides correct input / output mappings, while the network will have to adjust its free parameters in order to emulate the teacher in a statistically optimal way; • Unsupervised: the network learns independently from any ‘teacher’s’ intervention.

Let us see some existing types of learning.

1. **Learning with error correction.** Each neuron *k* receives an input of stimulus *x(n)* and generates a response $y_k(n)$, being *n* a discrete time. We also indicate with $d_k(n)$ the desired answer. As a result, an error signal $e_k(n)$ is generated.



   The error signal $e_k(n)$ implements a control mechanism with the aim of applying a sequence of adjustments to the synaptic loads of neuron k in order to bring the obtained response closer to the desired one.

2. **Memory-based learning.** In memory-based learning, all (or many) of past experiences are stored in a large memory of correctly classified input-output pairs. When the classification of an example never met before x_{test} is

requested, the system answers by finding and analyzing the stored examples surrounding x_{test}. All memory-based learning methods involve 2 basic ingredients:

- The criterion used to define the surrounding of a x_{test} test vector;
- The learning method applied on the examples surrounding x_{test}.

An example of this method is the **nearest neighbor method** in which the example closest to the test x'_N example is the one with minimal Euclidean distance.

$$\mathbf{x}'_N \in \{\mathbf{x}_1, \mathbf{x}_2, \ldots \mathbf{x}_N\}$$
$$\min_i [d(\mathbf{x}_i, \mathbf{x}_{test})] = d(\mathbf{x}'_N, \mathbf{x}_{test})$$

where *d* is the Euclidean distance. The class that is assigned to x_{test} is the same as for x'_N. A variant of this method is the k-nearest neighbor in which:

- the neighborhood of the test example is no longer one, but the set of *k* stored examples closer together;
- the assigned class is the one with the highest frequency surrounding the test example.

3. **Hebbian learning.** This is based on the Hebb postulate on learning, according to which when the axon of a neuron A (its output transmission line) is close enough to excite a neuron B and this, repetitively and persistently, sends an action potential, a growth process starts in one or both of the neurons which increases the efficiency of A. From this we can derive two rules:
   - if 2 neurons connected by a synapse are activated simultaneously, then the weight of the synapse is progressively increased;
   - If 2 connected neurons are activated asynchronously, then the weight of the synapse is progressively decreased or eliminated.

A synapse of this type is called the Hebbian synapse. If the correlation of the signals leads to an increase in synaptic efficacy we will call this a hebbian modification, if it leads to a reduction we will call it anti-hebbian.

• **Competitive learning**. With competitive learning, neurons from a neural network compete with each other to become active. Only a neuron can be active at a certain *n* time. There are 3 basic elements in a competitive learning method:

- a set of identical neurons, unless randomly generated synaptic weights respond differently to a given set of inputs;
- a limit to the "strength" of each neuron;
- a mechanism that allows neurons to compete for the right to respond to a given subset of inputs so that only one neuron or one group is active at a certain time. The winning neuron is called the **winner-takes-all**.

In this way neurons tend to specialize on a set of similar inputs and become specialized in recognizing the features of different classes of inputs. In the simplest form, a neural network has only one layer of output neurons, completely connected to the input knots (**forward excitatory connections**). The network may include connections between neurons that lead to lateral inhibitions (**inhibitory feedback connections**).

Advantages and disadvantages

So, we have seen an overview of the msin basic features of a Neural Network and understood how it basically works. I is also important that you are able to mentally locate Neural Networks in a broader framework, so we shall highlight what are their major advantages and disadvantages in the field of innovation. The advantages of using a NN are: • that they are suitable for problems that do not require accurate answers, but approximate answers with a degree of error or variation • their ability to generalize: they can produce good answers even with input that has not been considered during their creation and training • they are easy to implement, you can just define a neuron and then copy it and create connections between the neurons • it delivers fast operations because it works in parallel; every neuron uses only its input • the stability of the output with respect to input values: input values can be incomplete, noisy, not well known, or accept a degree of error or change • they can determine the result taking into account all inputs at the same time.

On the other hand, using Neural Networks entail a series of disadvantges: • a Neural Network works like a black box: generally, you will not be able to understand why it produced that specific result • the memorized knowledge can not be described and localized in the network • they must be used on serial computers, meaning there often will be a lack of hardware to implement them • they often entail sophisticated training techniques that take a long time for calculations • there is not always a network that can solve a specific problem, because there is not always a learning algorithm that converges giving a low error network output • the output values are not accurate, but have a margin which may vary • we need a very large series of examples to have a good learning process and a low output error.

Key points of this Chapter

- Artificial Neural Networks are able to make decisions independently, based on system inputs, which also include error variables. It is thanks to these that the systems based on neural networks are able to improve with recurring iterations, without human intervention;
- When we see a neural network, we find a numeric vector, which represents different types of data (pixels, audio signals, video signals or words, just as some possible examples). The input vector is generally transformed by a series of functions that work on the vector itself, and the result is the output generated by the network;
- The great peculiarity of neural networks is that the product of the system is the prediction of some properties that the network itself tries to guess starting from the input received. A trivial example can be the one in which we have an image when entering the neural network and, on the basis of some functions, it tries to guess if there are cars in the picture;
- Clearly, the functions of the system are managed by the ‘brain’, which in this case is a memory containing in turn other vectors of numbers known as "weights". The latter define how the network inputs must be combined and recombined to produce the most reliable result possible;
- The more complex the problem is, the more the weights have to become large (in informative terms). For example, recognizing a car inside an image can be a relatively difficult task, depending on the quality, the polygonal linearity or the spatiality of the objects in the picture;
- The most difficult task of a designer of a neural network is the definition of the weights, and of what "values" these must assume to ensure that the system does a good job when there is the need to generate a prediction.

Chapter 3. Deep Learning

Notes to this Chapter

The aim of this Chapter is to understand the basics of Deep Learning, one of the most interesting and ambitious fields of research currently under development, which finds various applications in real life.

Let us give a memory to our Neural Network

In a neural network, we will always get the same response when we feed in the same input values. A neural network per se has no memory. If we use programming terms, we can define it as a **Stateless Algorithm**. In many cases (like for the car price estimate), this is exactly what we need. However, a thing that this type of model cannot do is to give answers by relying on data that has been used in a different moment of time.

Imagine I handed you a keyboard and asked you to write a story. But before you start, I have to guess the first word you will digit. How can I guess what that word will be?

I can use my knowledge about language to increase my chances of guessing the right word. For example, you will probably use a word with which many stories start, like in “Once upon a time…”. If I could look at other stories that you have written in the past, I could narrow down my guesses to the words you usually use at the beginning of your stories. Once I have all this data, I could use it to build a neural network to know the probability with which you could use any given word. However, let us make the problem more difficult. Let us say I need to guess the next word you will be typing at any random place in your story. This is a much more interesting issue.

Let us use the first words in the book Pride and Prejudice by Jane Austen: “*It is a truth universally acknowledged that a single man in possession of a good fortune, must be in want of a*...”

What word would you write after these? You would probably say, “The word is probably destined to be ‘wife.” We know answers like this according to the letters we have already seen in the sentence and our common knowledge. In other words, it is easy to guess the next word if we consider the sequence of words that is written and our knowledge of the rules of language.

To solve this problem with a neural network, we need to add memory to our model. Whenever we ask our neural network for a response, we must also save a series of intermediate calculations that we can re-use the next time as part of our input. In this way, our model will also adjust its predictions on previously processed data.

Keeping track of status in our model makes it possible not only to predict the first most likely word in a story, but also to predict the most likely next word given all the previous words.

This is the basic idea of **Recurrent Neural Networks** (RNN): upgrading the network every time we use it. This allows you to update your predictions based on what you have experienced before. It can also create patterns over time, as long as it has enough memory.

While the ability to predict which the next word in a story may seem rather useless, think about the self-prediction function in your cell phone keyboard. And what if we bring this idea to the extreme? What if we ask the model to predict the next most likely word repeatedly - always? We are asking a computer to write an entire book for us!

The example of book writing Software

We saw how we could guess the next word in the previous paragraph. We can try to create a whole story using Jane Austin's style.

To do this, we will use the Implementation of a Recurrent Neural Network, written by Andrej Karpathy. Andrej is a researcher at Stanford and has written an excellent introduction to text generating with RNNs. You can see all the code for the model on **github**. We will create our model from the full text Pride and Prejudice, 120, 697 words of which 84 unique letters (including punctuation, uppercase, etc.). This data set is actually very small compared to applications in the real world. To really create a good model in Austin's style, it would be much better to have many more samples of text. However, this is enough for this example. Since we have just begun to implement it, our Recurrent Neural Network is still not very good at predicting the words. Here is what it generates after 100 cycles:

"singtunsIon posleged acessged ioledng oledged thatsessle mustsalust saalwa musanly ntallyive bebeall"

You can see that he understands that words have spaces between them, but that is all. After about 1000 iterations, things seem more promising: *"Niveally allal niveversmust waly ustust almu wanlying ortune, in change verally irtu of lywanter wunf"*

The model begins to identify basic structures such as phrases, added the punctuation at the ends of the sentences. A few words are recognizable and there are still a lot of meaningless words. But after several thousand more interactions, it seems that the result is much improved: *"It is a universal acknowledge that a man in single possession of fortune good, be in must want of a..."*

At this point, the algorithm understood the basic model used in Austin's writing rhythm. Some sentences almost make sense. Let us compare it with a true text of the book: Even with the use of a search pattern on a character at a time, our algorithm has reproduced a prose with a very plausible appearance and formatting. Unbelievable!

We must not completely generate the text from scratch. We can use the algorithm, providing the first words and leaving to the algorithm the further ones.

However, the amazing thing is that this algorithm can figure out the patterns on any data sequence. It can easily create a book that resembles a recipe or one of Obama's fake speeches. But why should we limit ourselves to human language? We can apply the same idea to any type of data that has a base model.

Deep learning: the ability of learning to learn

As we also learnt in the previous chapter, neural networks are implemented in machines capable of learning from wrong predictions. These in fact must be known (and therefore used as input) by the system, which must analyze, understand and correct its outputs. During the process of initializing a neural network, the values of the arrays constituting the weights are roughly assigned randomly, and then adjusted as the machine learns. In fact, the main goal of a neural network is to arrange, after iteration, the values of the weight, so that the next prediction is more precise than the one immediately preceding it!

That said, it may seem simple, but in reality it is not, especially in the most complicated scenarios. Training a neural network to learn is a very complex task, even more so if we think about areas such as speech recognition, computer vision or self-driving cars. Without any doubt, computational power is of primary importance for a quality neural network, and only in recent years, the use of graphics cards has allowed new results, exploiting them in parallel to significantly accelerate the process of predicting output.

Arrived at this point of the book, what do you think is the main criterion for teaching a machine to learn?

What a neural network must have in order to perform, is the analysis of the raw data deriving from the input. Once the input has been analyzed and packaged, neural networks are able to extract its characterizing properties, which are the basis for learning.

With the conventional approach to machine learning, these properties were identified manually, relying on recording in memory the characteristics of the input patterns: in essence, known inputs were fed into the neural network, and for each of these, the main properties were stored, making it a very long job. When the system received an input that had already been analyzed in the past, it recognized the similarity and was able to extract the characteristics of interest, as it was a case that it already had seen. Clearly, the more varied the inputs, the more flexible the neural network would become.

Deep learning removes the need for this type of human intervention and, therefore, the need to place known inputs in order to teach the machine to identify the most "frequent" properties.

In deep learning, increasing amounts of data are used to improve the ability of the neural networks to "think" and "learn" through the increased amount of processed data. "Deep" refers to the various levels of learning that the neural network accumulates over time, improving performance with the increasing of the network depth.
Although most of the current deep learning is performed under human supervision, the goal is to create neural networks that can train themselves and "learn" independently.

Deep learning is a relatively recent development, since only recently we have achieved the processing power and advanced data storage capacity that allows deep learning to be used to create new and exciting technologies.

This type of machine learning is the basis of "intelligent" technology, which ranges from speech/image recognition software to driverless cars. Advances in deep learning and robotics could soon lead to intelligent medical imaging technologies that can make reliable diagnoses, drive drones, and handle automated maintenance of machinery and infrastructure of all kinds.

The breakthrough innovation that deep learning brings is that of enabling machines to access second level-learning. We have taught computers to *learn how to learn*.

Let us understand what this means with a few examples. During his professional life, a radiologist examines thousands of radiographies, with which he acquires the necessary experience to say whether an X-ray represents a tumor initiation or not. While (unfortunately) a radiologist can make a mistake, a software that has examined millions (and not thousands) of chest radiographies could be able to give a much more precise response to the same problem.

If you think about working in the transport world, it is better to know that deep learning exists because in a few years cars could be driven by software, trained to recognize the edges of the road, the traffic lights, signs and obstacles in motion and with GPS they will surely be better than humans in safety performing on roads.

Deep learning could be used to build new anti-spam software that could be trained every day with millions of spam emails becoming better than any human does at recognizing a fake email.

Software that is able to read could become better than you could at writing summaries, press reviews and analysis of brand's reputation. Even now, translators, have a rather difficult future because machine translation software in the last ten years has almost equaled the capabilities of human beings.

Facebook already uses deep learning techniques for enabling computers to recognize a face in a photograph , so to suggest tags on an image); in every sector there are startups that are deepening the use of deep learning in a restricted area to solve specific problems, and by doing so, little by little, the revolution will soon or later be complete.

How does Deep Learning work

Deep Learningis an area of Machine Learning that is based on a particular type of data learning. It is characterized by the effort to create a model of machine learning on more than one level, in which the deeper levels take inputs from the previous levels, transforming them and abstracting them more and more. This insight into learning levels gives the name to the whole area (deep learning) and is inspired by the way the mammalian brain processes information and learns, responding to external stimuli.

Each level of learning corresponds, in this hypothetical parallel, to one of the different areas that make up the cerebral cortex. For example, the visual cortex, which is responsible for the recognition of images, shows a sequence of sectors, placed in hierarchy. Each of these sectors receives an input representation, by means of the stream signals that connect it to the other sectors. Each level of this hierarchy represents a different level of abstraction, with the most abstract features defined in terms of those of the lower level. When the brain receives images, it processes them through different phases, for example the detection of the edges, the perception of the shapes (from the primitive ones to the gradually more and more complex ones). This is why we speak of hierarchical representation of the image at the level of increasing abstraction.

“What??” your granny would say. Let me explain the concept with different words.

As the brain learns by trial and error and activates new neurons learning from experience, even in the architectures responsible for Deep Learning, the stages of extraction can be modified according to the information received at the entrance.

Neural Networks are one of the main tools that benefit from Deep Learning. In fact, even if a three-layer network (with one input layer, one hidden layer and one output layer) does have a certain degree of ability in distinguishing between arbitrarily complex regions, a larger architecture brings some advantages.

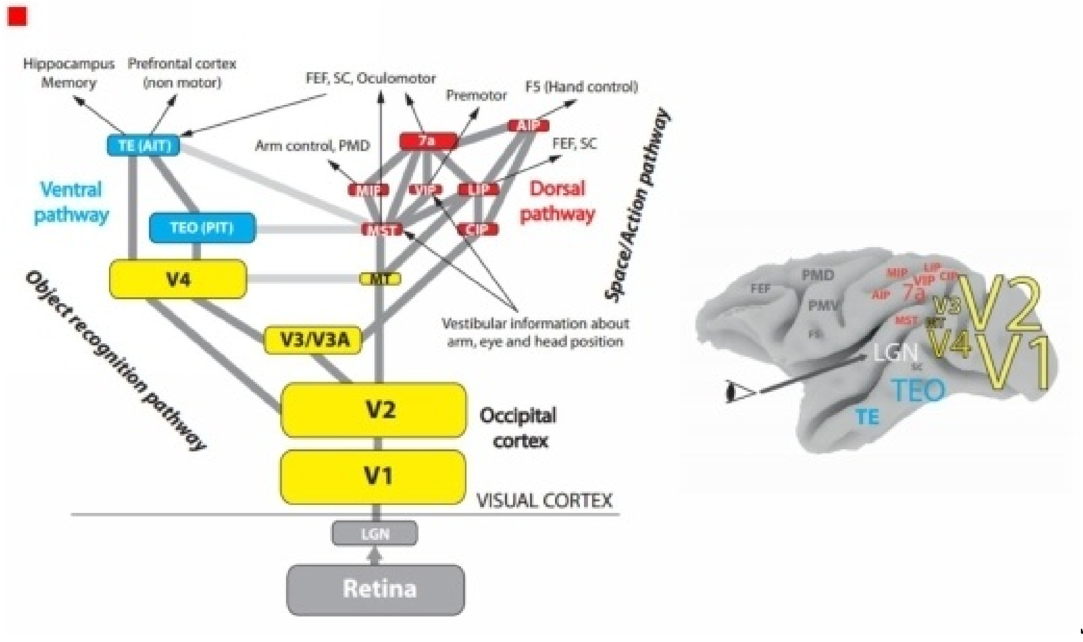
One of the main advantages, for example, is related to the number of knots per layer: in fact, in a three-level network, the number of knots in each layer limits the complexity of the region to be recognized. As a result, we risk, by limiting the depth, to be in need a large number of knots. Theoretical results show that there are situations in which the number of necessary knots increases exponentially with the size of the input. An explosion of knots on the layers causes a high computational cost and a huge use of memory: for this reason, a careful planning of the network architecture to be used must be always carried out.

Main architectures and algorithms

Deep Neural Networks The term DNN (Deep Neural Network) indicates "deep" networks composed by many layers (at least two of which are hidden) organized hierarchically. Hierarchical organization allows to share and reuse information (a bit like structured programming). Along the hierarchy you can select specific features and discard unnecessary details (in order to maximize the invariance). The most simple structure we know is that of **feed forward networks** trained through **back-propagation algorithms**.

The updating of the weights during the learning phase is carried out using the SGD (Stochastic Gradient Descent) method, that is in formulas:

$$w_{ij}(t+1) = w_{ij}(t) + \eta \frac{\partial C}{\partial w_{ij}}.$$



Just as an example, the human body's visual system operates on a hierarchy of levels (deep): The most widely used DNNs consist of a number of layers included between 7 and 50. Deeper networks (100 levels and above) have shown they can guarantee slightly better performance, but at the expense of efficiency. The depth (number of levels) is only one of the factors of complexity: the number of neurons, connections and weights also characterize the complexity of a DNN. The greater the number of weights is (i.e., the parameters to be learned) the greater will be the complexity of the training. At the same time, a high number of neurons (and connections) makes forward and back propagation more expensive.

These networks are used in particular in language modeling, object recognition and more generally for modeling complex non-linear relationships. Some limits of this type of network are the high execution time of the learning phase and the risk of overfitting, caused by the possibility for networks as deep as these to specialize on the specific features of the training set, losing their ability to generalize.

Main types of DNN

The main types of Deep Neural Network are:

1. **"Discriminatory" feedforward models** for classification (or regression) with predominantly supervised training:
   - CNN - Convolutional Neural Network (or ConvNet)
   - FC DNN - Fully Connected DNN (MLP with at least two hidden levels)
   - HTM - Hierarchical Temporal Memory
2. **Unsupervised training** ("generative" models trained to reconstruct the input, useful for pre-training of other models and for producing salient features):
   - Stacked Auto-Encoders
   - RBM - Restricted Boltzmann Machine
   - DBN - Deep Belief Networks
3. **Recurring models** (used for sequences, speech recognition, sentiment analysis, natural language processing, ...):
   - RNN - Recurrent Neural Network
   - LSTM - Long Short-Term Memory
4. **Reinforcement learning** (to learn behaviors):
   - Deep Q-Learning

Convolutional Neural Networks CNNs are a development of deep neural networks, that, being designed specifically for image recognition, use a particular architecture. Each image used in learning is divided into topologically compact portions (portions which properties of space that are preserved even when deformed), each of which will be processed by others to search for particular patterns. Formally, each image is represented as a three-dimensional array of pixels (width, height, color) and each of its sub-sections is put in convolution (a mathematical way of combining two signals to form a third signal) with the chosen filter. In other words, sliding each filter along the image, the internal product between the filter itself and the input is calculated. This procedure produces a set of feature maps (**activation maps**) for the various filters. Overlaying the various feature maps of the same portion of image we get an output volume. This type of layer is called **convolutional layer**. After the convolutional layer we have a **sub-sampling layer**, that is a layer that deals with further subdivision of the processed image, in such a way as to analyze in detail its sub-sections, reducing its dimensionality. One type of sub-sampling among the most used is the **MaxPooling** sub-sampling. In particular, this type of sampling partitions the image into a series of rectangles that are not overlapped, and returns the pixel from each rectangle corresponding to the maximum value point. These two types of layers will alternate throughout the body of the network. It is also possible to find other types of layers combined with convolution and pooling layers. The last layer is the output layer: it has as many neurons as possible labels exist and for each label it gives the probability that the tested sample may belong to it.

The training phase starts by processing a group (batch) of training examples at a time and, for each of them, after obtaining the output, it calculates the cost

$$L_i = -\sum_j t_{i,j} log(p_{i,j})$$

functions and the cross-entropy:

This is minimized by means of algorithms such as the SGD (**Stochastic Gradient Descent**) method and used for back propagation. All of the following Chapter will be dedicated to learning the details of Convolutional Neural Networks.

Stacked Auto-encoders Stacked Auto-encoders are deep neural networks that are used purely for data compression. Their specific "hourglass" structure clearly shows a first part of the process in which the input data is compressed, to arrive at the so-called "bottleneck", from which the decompression starts again. The output is hence an approximation of the input. These networks are unsupervised in the pre-training phase (compression) while the fine-tuning phase (decompression) is supervised. The particularity of Stacked Auto-encoders is that of being composed by a series of classic encoders connected one after the other.

An interesting variant is that of the **Stacked Denoising Auto-encoders**, in which the output is cleaned up so as to try to further lower the noise obtained during compression, and to make the algorithm more robust. In this way, the network can also be used for the reconstruction of corrupted inputs.

Available Frameworks and libraries

This section presents some of the most used frameworks for Deep Learning. In summary, almost all libraries provide the possibility to use the processor in order to speed up the learning process, are released with open license and are the result of university groups' research. Let us see the main ones:

- **Theano**. Theano is probably the used library, being written in Python, one of the most used languages in the Machine Learning field. It also allows calculations to be carried out through the GPU, getting even 24 times better performance than when using the CPU. It allows defining, optimizing and evaluating complex mathematical expressions such as multidimensional arrays. Among its declared features there are:
    - Integration with the NumPy library;
    - An efficient differential symbolic calculus;
    - Optimization at the level of speed and stability;
    - Dynamic generation of C code.
- **Caffe**. Developed primarily by Berkeley Vision and Learning Center (BVLC), it is a framework designed to stand out for its expression ability, speed and modularity. Its particular architecture encourages application and innovation and it allows to pass easily from calculations on the CPU to calculations on the GPU. The rather large community of users has allowed considerable development in recent times. It is

written in Python language but the installation process can be long due to the numerous support libraries to be compiled.

- **Lasagne** is a Python library that relies on Theano, designed to easily implement neural networks for Deep Learning at a higher level of abstraction. The project is still in a phase of strong expansion and has one of the best documentations available. The installation is simple and retains all Theano's strengths as the possibility of using the processor to increase computing performance.
- **Deeplearning4j.**Written for Java and Scala languages, Deeplearning4j is the first library - among those analyzed - designed for commercial purposes, in fact it can interface with Hadoop and Spark. It contains numerous techniques and is released in an open source way.
- **Torch.** Torch is a vast machine learning ecosystem that has a large number of algorithms and functions, including deep learning and processing of various types of multimedia audio and video data, with a special focus on parallel calculus. It provides an excellent interface for C language and has a large user community.

Key points of this Chapter

- Deep Learning is a set of statistical algorithms and techniques that allow you to find patterns, recurring schemes, and regularities (or irregularities) in an unorganized data set;
- DL actually abates costs, since up to now the most resource consuming activity when building a neural network was having to feed the computer with thousands of inputs (the images of a dog, for example), in order to enable it to recognize one. Through deep learning it is the computer itself that, being fed automatically with millions of raw data (marked or not marked), learns and finds the patterns that identify an object (as the image of a dog);
- A DNN is a neural network that has more hidden layers between input

and output levels;

- Theano, Caffe, Lasagne, Torch and Deeplearning4j are the main libraries that can be used as valuable frameworks for Deep Learning implementations.

Chapter 4. Convolutional Neural Networks

Notes to this Chapter

The aim of this Chapter is to dive into the detail of one of most fascinating and promising types of Deep Neural Network: CNN - Convolutional Neural Networks.

Deep Learning and Convolutional Neural Networks (CNNs)

In this Chapter, we will learn to use Deep Learning to write a Convolutional Neural Network that recognize objects within images. In other words, we are going to explain the black magic that allows Google Photos to look for photos based on what is in the photo and not by title or tag.

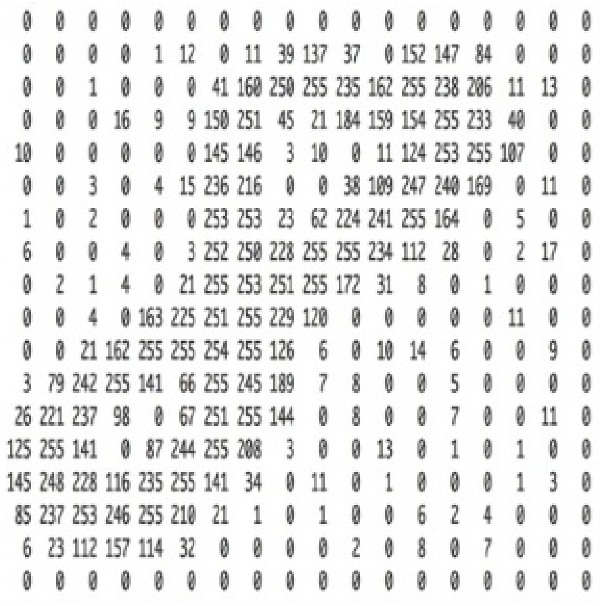
Every 3-year-old baby can recognize the photo of a bird, but understanding how to reproduce the ability of recognizing objects on a computer has puzzled the best computer scientists for over 50 years.

In recent years, we have finally found a good approach to detecting objects using **Deep Convolutional Neural Networks**. Let us try to write a program that recognizes birds in images.

First, we must start from the basics. Before learning how to recognize bird images, let us try with the task of recognizing something much simpler - a "2" handwritten number. In the Chapters of this book, we learned how neural networks can solve complex problems by concatenating a series of simple neurons. With this approach, we created a small neural network to estimate the price of a car based on its min relevant features; we also know that the same general machine learning algorithms can be reused with different data to solve other problems. So let us try to modify that same neural network we used for cars for the recognition of handwritten text.

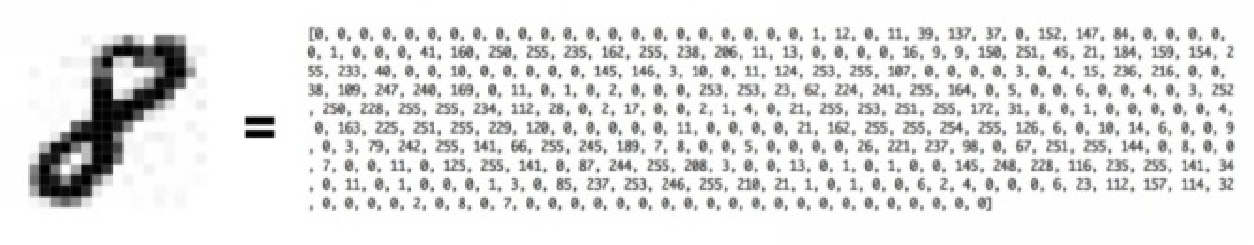
To make the task very simple, we will only try to recognize one figure - the number "2". Machine Learning works only when you have data - preferably a large amount of data. So we need a large number of handwritten "2"s. Fortunately, some researchers have created the **MNIST dataset** of handwritten numbers, a series of handwritten numbers precisely for this purpose. MNIST provides 60,000 images of handwritten digits, each as 18x18 image. Here are some "2"s taken from the MNIST dataset:

The Neural Network we built in the previous Chapters was able to take only a fewvalues as input (power of the motor, mileage, etc.). But now we want to use our Neural Network to process images. How do we feed input to our neural network using images and not numbers? The answer is very simple. A neural network only takes numbers as inputs. For a computer, an image is nothing more than a grid of numbers that represent the degree of darkness of each pixel:



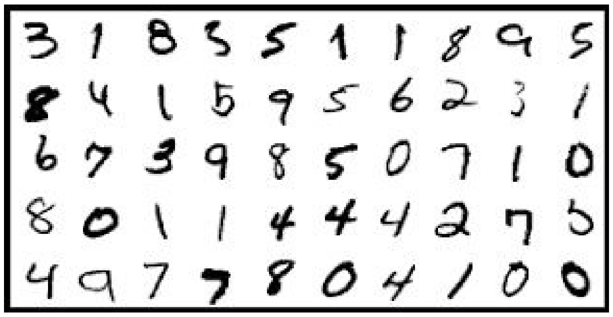
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	12	0	11	39	137	37	0	152	147	84	0	0	0
0	0	1	0	0	0	41	160	250	255	235	162	255	238	206	11	13	0
0	0	0	16	9	9	150	251	45	21	184	159	154	255	233	40	0	0
10	0	0	0	0	0	145	146	3	10	0	11	124	253	255	107	0	0
0	0	3	0	4	15	236	216	0	0	38	109	247	240	169	0	11	0
1	0	2	0	0	0	253	253	23	62	224	241	255	164	0	5	0	0
6	0	0	4	0	3	252	250	228	255	255	234	112	28	0	2	17	0
0	2	1	4	0	21	255	253	251	255	172	31	8	0	1	0	0	0
0	0	4	0	163	225	251	255	229	120	0	0	0	0	0	11	0	0
0	0	21	162	255	255	254	255	126	6	0	10	14	6	0	0	9	0
3	79	242	255	141	66	255	245	189	7	8	0	0	5	0	0	0	0
26	221	237	98	0	67	251	255	144	0	8	0	0	7	0	0	11	0
125	255	141	0	87	244	255	208	3	0	0	13	0	1	0	1	0	0
145	248	228	116	235	255	141	34	0	11	0	1	0	0	0	1	3	0
85	237	253	246	255	210	21	1	0	1	0	0	6	2	4	0	0	0
6	23	112	157	114	32	0	0	0	0	2	0	8	0	7	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

To feed an image in our neural network, we must transform the 18x18 pixel image into a matrix of 324 numbers (below):



```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 12, 0, 11, 39, 137, 37, 0, 152, 147, 84, 0, 0, 0, 0,
0, 1, 0, 0, 0, 41, 160, 250, 255, 235, 162, 255, 238, 206, 11, 13, 0, 0, 0, 0, 16, 9, 9, 150, 251, 45, 21, 184, 159, 154, 2
55, 233, 40, 0, 0, 10, 0, 0, 0, 0, 0, 145, 146, 3, 10, 0, 11, 124, 253, 255, 107, 0, 0, 0, 0, 3, 0, 4, 15, 236, 216, 0, 0,
38, 109, 247, 240, 169, 0, 11, 0, 1, 0, 2, 0, 0, 0, 253, 253, 23, 62, 224, 241, 255, 164, 0, 5, 0, 0, 6, 0, 0, 4, 0, 3, 252
, 250, 228, 255, 255, 234, 112, 28, 0, 2, 17, 0, 0, 2, 1, 4, 0, 21, 255, 253, 251, 255, 172, 31, 8, 0, 1, 0, 0, 0, 0, 0, 4,
 0, 163, 225, 251, 255, 229, 120, 0, 0, 0, 0, 0, 11, 0, 0, 0, 0, 21, 162, 255, 255, 254, 255, 126, 6, 0, 10, 14, 6, 0, 0, 9
, 0, 3, 79, 242, 255, 141, 66, 255, 245, 189, 7, 8, 0, 0, 5, 0, 0, 0, 0, 26, 221, 237, 98, 0, 67, 251, 255, 144, 0, 8, 0, 0
, 7, 0, 0, 11, 0, 125, 255, 141, 0, 87, 244, 255, 208, 3, 0, 0, 13, 0, 1, 0, 1, 0, 0, 145, 248, 228, 116, 235, 255, 141, 34
, 0, 11, 0, 1, 0, 0, 0, 1, 3, 0, 85, 237, 253, 246, 255, 210, 21, 1, 0, 1, 0, 0, 6, 2, 4, 0, 0, 0, 6, 23, 112, 157, 114, 32
, 0, 0, 0, 0, 2, 0, 8, 0, 7, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

To manage 324 inputs, it is sufficient to expand the Neural Network to 324 input knots. Note that now our Neural Network also has two outputs (instead of one). The first output will calculate the probability that the image actually is a "2", while the second output will calculate the probability that the image provided is not a "2". Having a separate output for each type of object we want to recognize, we are now able to use a neural network to classify objects in groups. Our neural network is much bigger than the last time (324 inputs instead of just a few!). However, any modern computer can handle a neural network with a few hundred knots without blinking. This Neural Network could work smoothly even on your cell phone. All that remains to be done is to train the neural network with images of "2"s and “not-2"s to make sure that it learns to distinguish them. When we insert a "2", we will say that the probability that the image is a "2" is 100%, and the probability that it is a “not-2" is 0%. Conversely for counter-example images. Here are some examples of our training data:



On a modern laptop computer, we can train this kind of neural network in a matter of minutes. After completing the training, we will have a neural network capable of recognizing "2" images with very high precision. Welcome to the world of image recognition! (At least as it was done at the end of the 1980s).

Tunnel Vision

It is surprising that you just need to feed pixels into a neural network to teach it how to recognize images. All this is fantasy! …. right?

Wrong, things really are not that simple.

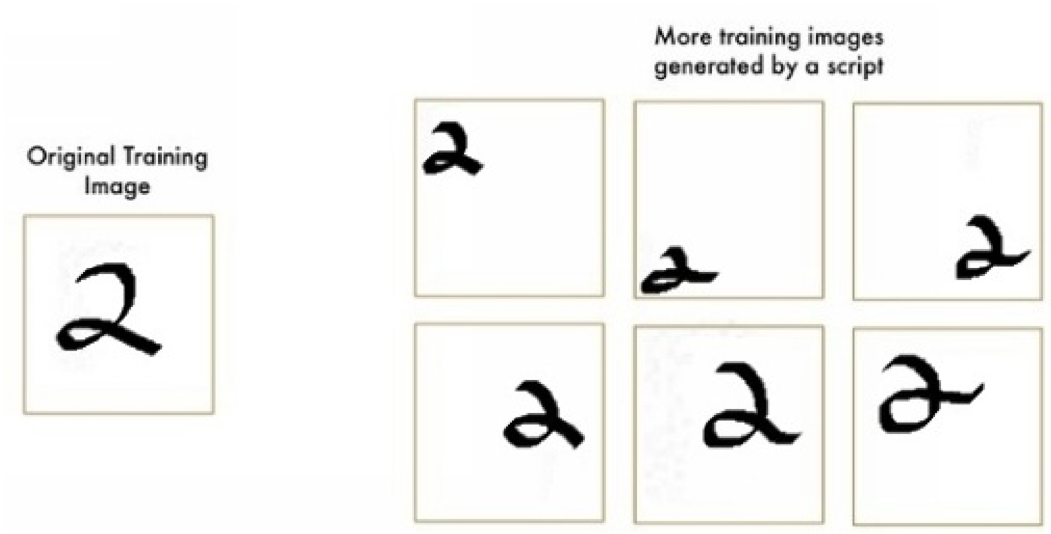
First, our "2" recognition algorithm works well with simple images where the letter lies perfectly in the center of the image. The bad news is that our "2" recognition algorithm completely stops working when the letter is not in the exact center of the image. Just the slightest change in position can ruin everything. This is because the neural network only learns to recognize a perfectly centered version of a "2", but has no idea of what a "2" out of the center looks like. The neural network knows only one version and one pattern.

This is not very helpful. Problems in the real world are never so simple and clean. Therefore, we need to find a way to run the neural network in cases where the "2"s are not perfectly centered.

So, we have already created a great program to identify a "2" in the center of the image. And if we made a series of scans in smaller sections of the image looking for other "2"s until we find one, would it work? This approach is called the **Sliding Window** approach. It works well in some limited cases, but it is really inefficient. You control the same image multiple times looking for objects of different sizes. We can do much better than that!

To train the neural network we used a series of perfectly centered “2”s, but what would happen if we train it with more data, including "2"s of various sizes and scattered across the image?

We do not even need to collect new training data. We can use a script to generate new images with "2"s in all possible positions within the image:



We have created synthesized training data by creating different versions of training images from the one that we already had. This is a very useful technique! By using this technique, you can easily create an infinite stock of training data. More data makes the problem more difficult to solve for our neural network, but we can overcome this problem by increasing its size to make it able to handle much more complex patterns. To make the network bigger, just add layers to the existing layers of knots. This architecture is an example of Deep Neural Network with multiple layers. As we know, we must use our 3D graphics cards (which is designed to make extremely fast multiplications of matrixes) instead of normal computer processors; nevertheless, even if we are able to enlarge our Neural Network and train it quickly with a 3D graphics card, we remain far from solving the problem of image recognition. We need to use a smarter approach to process images in our neural network. Think about it. It does not make sense to train a neural network to separately recognize a "2" at the top of an image and a "2" at the bottom of the same image. We would need a way to make the neural network smart enough to know that a "2" is the same regardless of in which part of the image it is. Luckily ...there is!

Convolution

For all humans it is intuitive to know that images have a hierarchy or conceptual structure. Consider this image:

A human being is able to immediately recognize the hierarchy in this image:

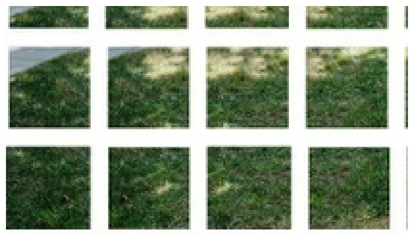
- The ground is covered with grass and cement
- There are 3 puppies
- The puppies are laying on the cement

More importantly, we recognize the idea of a dog regardless of what the surface is upon which it is laying. We did not have to learn the dog idea repeatedly for any possible surface on which it might appear. However, for the time being, our neural network cannot do this because it considers that a "2" in a different part of the picture is a completely different thing. It does not understand that moving an object inside the photo does not make it something different. This means that it must re-learn the identity of each object in all of the positions. A nice problem!

We must ensure that our Neural Network is able to interpret the **Invariance Translation** that is, a "2" remains a "2" regardless of in which portion of the photo you are. We will do it using a process called **Convolution**. The idea of Convolution is partly inspired by computer science and partly by biology.

Rather than fueling entire images in our neural network as a grid of numbers, we are going to do something much smarter that exploits the idea that an object remains the same, no matter where it is inside the photo. Here is how we are going to work, step by step:

- Step 1: Break up the image in small tiles



As for the Slide Window solution described above, we pass a sliding window across the original image and save each result as a small separate tile: In this way, we transformed our original image into *X* number of smaller images of equal size.

- Step 2: Insert each tile into a small Neural Network

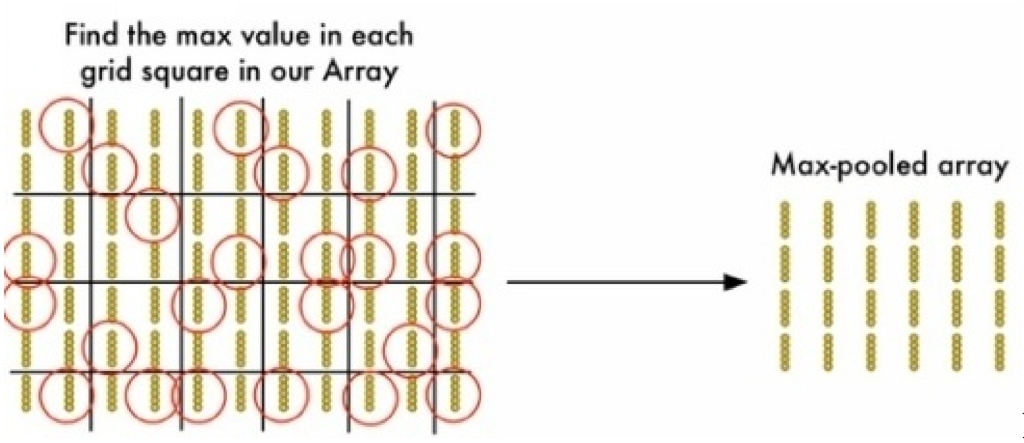
Previously, we fed a single image in our neural network to see if it the image contained a "2" or not. We will do the same thing here, but we will do it for each of the *X* number of tiles created. However, there is a big difference in this case: we will keep the same weight for each tile respect to the original image. In other words, we are handling each set of images in the same way. If something interesting appears on a specific tile, we will only take note of that tile as particularly interesting.

- Step 3: Save the results of each tile to a new array

To keep track of the original position of the tiles we will save the result of our processing in a grid, maintaining the same arrangement as the tiles in the original image. In other words, we started with a large image and we ended up with a slightly smaller table that contains information about which tiles of the original image that the neural network considers interesting.

- Step 4: Downsampling

The result of step 3 is a matrix that outlines which parts of the original image are the most interesting. However, this matrix still is quite large. To reduce the size of the array, we will use the **Max Pooling** algorithm we spoke about in the previous Chapters.

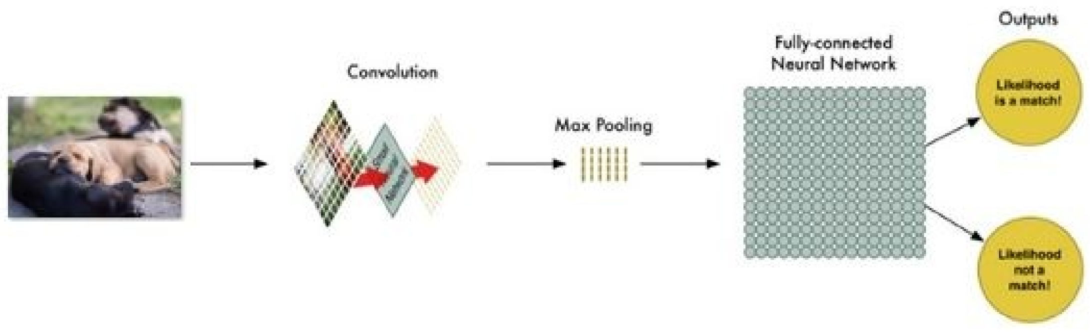


For each 2x2 square tile of the matrix, we keep only the highest values and get rid of everything else.

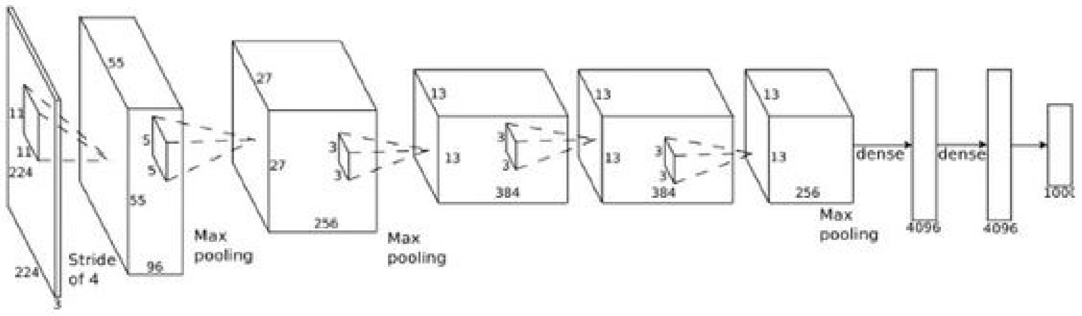
The idea is that if we find something interesting in one of the four input cards that make up each square of the 2x2 grid, we will just keep the most interesting pieces. This reduces the size of an array retaining the most important parts.

- Final step: Make a prediction

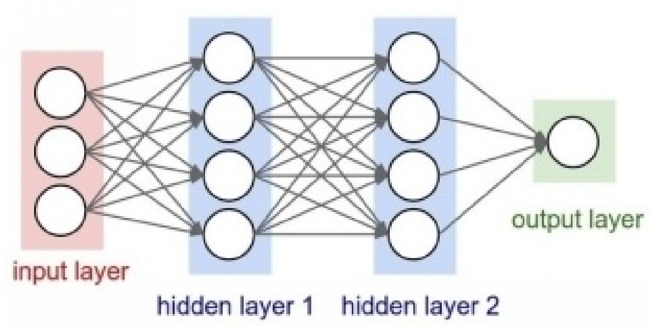
So far, we have reduced a large image to a much smaller size matrix. Guess what? This matrix is nothing more than a set of numbers that can be used as inputs in another Neural Network. This second Neural Network will decide whether the image is what we are looking for. To differentiate it from the previous Convolutional Neural Network, we will call it Fully Connected Neural Network. To wrap up, our Neural Network for Image Recognition will consist of 5 sub-networks and the final architecture will look like this:

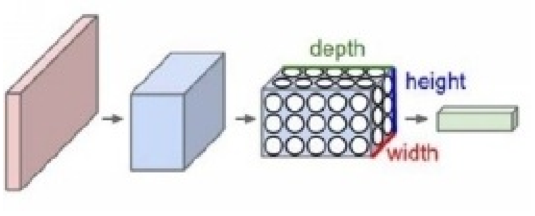


Our image processing pipeline involves a series of different steps: Convolution, Max-Pooling and finally a Fully Connected Neural Network. To solve real-world problems, these steps can be combined and stacked every time you think it is necessary! We can have two, three or even ten layers of Convolution. You can introduce all the Max-Pooling you want to reduce the size of the data. The basic idea is to start with a great image and continuously reduce the data step by step until you get a result. The more Convolution steps you have, the more your neural network will be able to understand and process complex functions. For example, the first step of a Convolution process can learn to recognize sharp edges, the second step to recognize a bird’s beak starting from the data on sharp edges, the third step could recognize whole birds using the knowledge about the beak structure, etc. Here is a Deep Convolutional Neural Network (as you might find in an academic article):



In the figure below, on the left you can see a fully connected Artificial Neural

Network. On the right a CNN: 



In the scheme we described in our example above, we start with a 224 x 224 pixel image, with two Convolution and Max-Pooling processes, followed by three Convolution processes, another Max-Pooling process, and two layers of Fully Connected Neural Networks.

The end result is that the Neural Network can place the image in the correct category among the 1000 possible!

The right Architecture for a Neural Network

How can you know what steps need to be combined in order to make sure the Neural Network is able to recognize the images? Honestly, the only way to give a unique answer to this question is by doing a lot of testing. You might need to test 100 networks before finding the optimal structure and parameters for the problem you are trying to solve. Machine Learning requires a lot of trial and error. What if we were to build now a bird classifier?

At this point we have learned enough about Deep Convolutional Neural Networks to be able to write a program that recognizes whether a photo contains a bird or not. As always, we need some data to get started. The CIFAR10 dataset contains 6,000 bird pictures and 52,000 images of things that are not birds. Nevertheless, to get even more data, we can add the Caltech-UCSD Birds-200-2011 dataset that contains more than 12,000 bird photos. Here are some of the bird's photos taken from the datasets described above:

And here we have some of the 52,000 photos that do not contain birds: This data set works well for our purposes, but 72,000 low resolution images would not be enough for real-world applications. To achieve performance at the same levels as

Google, we would need millions of high resolution images. In Machine Learning, having more data is usually more important than having better algorithms. That is why Google is so happy to offer unlimited space for your photos and documents. They want your data, sweet data! To build our classifier, we will use **TFLearn**. TFlearn is a Wrapper around the **TensorFlow Deep Learning Library** offering free of charge by Google, which exposes a simplified API precisely for this reason: making Convolutional Neural Networks easy to build as a couple of rows of code.

If we train our Neural Network with a good video card and enough RAM (such as an Nvidia GeForce GTX 980 Ti or better), the whole process will last less than an hour. If you train it with a normal CPU, it may take a lot longer. Increased training corresponds to an increase in identification skills. After the first passage, I got 75.4% of accuracy. After only 10 passages, we are already at a 91.7%. After 50 passages, the marginal precision increase reduces after every step to reach the precision peak of 95.5%.

Congratulations! Our program is now able to recognize bird pictures!

Test your Neural Network

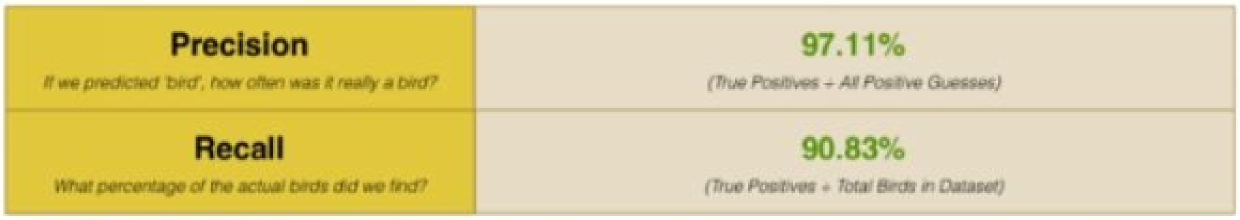
Now that we have a well-trained neural network, it is time to test it! Here is a simple script that given a single image is able to recognize whether it is a bird or not. To see how effective our network is, we need to try it with many images. For this purpose, I kept 15,000 images from the initial dataset to use as validation. When I analyzed those 15,000 images, the Neural Network found the correct answer in the 95% of cases. This seems a good result, right? Well, it depends, how accurate is 95% accurateness?

Our network claims to be correct in 95% of cases. However, the devil is in the details, and our 95% could mean many different things. For example, what if only 5% of our validation images contained birds while the remaining 95% did not? A program that predicted "It Is Not a Bird" every time would be accurate in 95% of cases! However, it would also be 100% useless. We need to look more closely at the numbers going beyond the overall level of precision. To judge the reliability of a classification system, we need to look beyond the error rate as well as the type of mistakes made. Instead of thinking about the predictions given by the Neural Network in terms of "right" and "wrong", we will divide them into four distinct categories:

1. True Positives: Birds Our Neural Network Identifies Properly as Birds.
2. Negative Positives: Photos That Do not Contain Birds and Our Neural Network Identifies Properly As Non-Birds.
3. False Positives: images that the Neural Network classifies as birds but that actually do not contain any birds. Many planes were confused for birds, understandable!
4. False Negatives: images that the Neural Network classifies as non-birds but which are actually birds.

Why should we break down the results this way? Because not all errors are the same. Imagine if we were writing a program to detect cancer though magnetic resonance imaging. In this case, we would prefer finding both false positives and false negatives. False negatives would be the worst case possible - the program would not be able to diagnose cancer in the case of a patient who is actually suffering from it delaying the starting of cures.

Instead of just looking at overall precision, we can calculate **Precision** and **Recall metrics** because they give us a clearer picture of the Neural Network's performance:



Precision *If we predicted 'bird', how often was it really a bird?*	**97.11%** *(True Positives ÷ All Positive Guesses)*
Recall *What percentage of the actual birds did we find?*	**90.83%** *(True Positives ÷ Total Birds in Dataset)*

This tells us that 97% of the time we have been able to identify the birds correctly!

But it also tells us that we found only 90% of the birds in the dataset. In other words, the Neural Network has not been able to recognize all of the birds, but it is very sure of its judgment once it has identified one!

Now that you know the basics of **Deep Convolutional Neural Networks**, you can try playing with **tflearn** and testing various Neural Network architectures. Tflearn provides data sets to skip the data collection step and immediately get to the algorithm writing part.

Key points of this Chapter

- In this Chapter, we have learnt what is the process to write a Convolutional Neural Network that recognize objects within images;
- We have learnt what Convolution is, a tool that resembles human intuitive understanding of the conceptual hierarchy within an image;
- We have gone through an image processing pipeline which involved a series of different steps: Convolution, Max-Pooling and finally a Fully Connected Neural Network;
- To solve real-world problems, these steps can be combined and stacked every time you think it is necessary. The more Convolution steps you have, the more your neural network will be able to understand and process complex functions.

CONCLUSIONS

"Torture the data, and it will confess to anything."

– *Ronald Coase* Machine Learning is the science that allows computers to perform certain actions without having been explicitly programmed to execute them and represents one of the fundamental areas of development in the future of artificial intelligence. Deep Learning is a branch of Machine Learning and is based on a set of algorithms that attempt to model high abstractions level in data; generally, these algorithms provide multiple stages of processing, having often a complex structure and normally these stages are composed of a series of non-linear transformations. As you have learnt in this book, convolutions are widely used in Deep Learning, especially for computer vision applications. The architectures used are often anything but simple.

Very recently, Facebook has decided to invest heavily in this sector with the aim of being able to understand (and exploit economically) the hidden meaning of every single post or image published by every one of the millions of users of the most famous social network of the world. All of the great hi-tech players, starting from the already mentioned Facebook up to Google, passing through Yahoo! and Microsoft are watching with great attention the developments that are taking place in this sector. Investing, perhaps, more than a few dollars in the most advanced research institutes. And the reason is soon said. Deep learning could (and should) improve the way computer systems analyze natural language. As a result, it should improve understanding. If the path taken will lead to the desired results, deep learning should allow the neural networks that form the information systems to process natural languagesjust as it happens in the human brain. Computers, in short, will be able to understand what human users write on their bulletin board or what they really want to look for; whether they are sad or

happy; if the image they have just viewed was disliked or not.
Facebook, Google, Microsoft and Yahoo!, therefore, could create increasingly in-depth and accurate commercial insights, reselling this data to communication and marketing agencies all over the world for increasingly accurate targeted advertising campaigns tailored on the needs of users.
As often happens in the world of high technology, also small companies have a pivotal role in launching and carrying forward great revolutions. And this has happened also in the field of deep learning, where small startups are doing a lot of the work. AlchemyAPI, for example, is a small software house that has been active in this sector for some time now. So far, its efforts have focused on the implementations for natural language recognition, but in recent months it has launched itsplatform of neural networks for image recognition.
Microsoft, however, is among the companies to have reported the greatest results in this field. Last November, at an event in China, Microsoft research labs showed the world what their deep learning systems are capable of, showing an automated system of instant translation from English to Mandarin Chinese. A bit underhanded, but also Yahoo! is trying to make progress in the sector. Following the acquisition and incorporation policy desired by CEO Marissa Mayer, Yahoo! has recently acquired two of the best start-ups in the field of deep learning: IQ Engines and LookFlow. Probably much of the know-how of these two companies will be used to make more and more "intelligent" and dynamic Flickr, but it is not excluded that the volcanic CEO can hold some surprises for its users.
Finally, Google is probably the company that has invested most in this sector to make its search algorithm perfect. Thanks to the steps forward recorded by deep learning, Big G is able to offer exclusive services and tools to its users. The voice recognition system on which *Google Now* is based, for example, is the result of in-depth research work in this area, as is the face recognition system on which one of the many photographic features of Google+ is based. Not to mention, then, the linguistic analysis tools used by the search engine, for results that are increasingly refined and close to the needs of users.

In Deep Learning there are still many unknown areas, and still a lot of implementations are needed. The theory that explains because it works so well is currently still incomplete and probably there is no book or guide that is better than direct experience.