An introduction to Artificial Intelligence and Deep Learning

Fabio Grazioso*
1) Microfiltration Processes Laboratory, WCRC "Advanced Digital Technologies", Tyumen State University, Volodarskogo 6, Tyumen, 625003, Russia;
2) Photonics and Microfluidics Lab, Tyumen State University, Volodarskogo 6, Tyumen, 625003, Russia;
3) Tyumen State Medical University, Odesskaya 54, Tyumen, 625023, Russia.

(Dated: November 20, 2022)

The present work introduces the main concepts from the research on Artificial Neural Networks (ANNs) which represent the most promising model to realize artificial intelligence. We will see how historically the main ideas have been developed, the periods of great development of the neural computational paradigm and its long period of oblivion. We will describe the main features of an ANN discussing in some detail an example of an application to optical characters recognition. We will follow how the technique of *backpropagation*, although a mere technical tool, has made possible the use of more powerful networks, bringing the times of computation down to acceptable values for practical applications. We will also describe in some details the more advanced and recent model of ANN, the Convolutional Neural Network.

CONTENTS

I. INTRODUCTION

Since the term *Artificial Intelligence* (AI), has become part of popular culture, it may have become too wide to be used in a scientific context.

In general we may say that AI refers to the study of *cognitive capabilities* shown by man-made artifacts. Those capabilities can be divided in the following categories: (a) the ability to *learn new behaviours*, not previously programmed in its design, (b) the ability for *proactive interaction* with an unknown environment, (c) the ability to infer and deduce new information.

A very limited list of the application fields can be: computer vision, speech recognition, problem solving, knowledge representation.

The implicit definition of AI on which the above statements are made is the *Strong AI*, as defined by Searle [1, 2]. In extreme summary, this definition of AI says that the artificial device that shows cognitive capabilities, is therefore assumed to have cognitive states, *intelligence*, self-awareness, or consciousness. This can be related to the Turing test [3] .

Among several authors who wrote critically on the subject of strong AI we can mention Roger Penrose [4]

The quest for an "intelligent artificial device" can be traced very far back in time, as back as the mentions to intelligent machines found in the bible, or in the greek mythology.

In the modern development of this scientific endeavour, there has been a time, between the two world wars, when two approaches have been compared to each other.

On one side, there was the idea of mimicking the structures that the research in biology and anatomy was discovering inside the brain and the nervous systems.

This approach may dated back 1943, with the seminal article by Mc Culloch and Pitts [5].

At the same time, another approach was being developed, relying on a much more abstract model, and on the idea of *algorithm*. A powerful thrust to this idea of the algorithmic approach to intelligence was coming from the famous *entscheidungsproblem*, the "decision problem" formalized by Hilbert in 1928 [6]. This problem can be loosely described as whether it is possible or not to decide *automatically* about the truth or falseness of any possible statement of formal logic. Turing, and Church, independently tackled the problem, by first giving a precise definition of "automatic decision", i.e. a precise definition of *algorithm* [7, 8].

* f.grazioso@utmn.ru

To trace the history of those two approaches is complex, and the von Neumann symbolic and algorithmic approach has gained the upper hand for a long period. The other approach, the neural approach, also called connectivistic, has gone through a long period of oblivion after the appearance of the work of Minsky and Papert, who formalized and made it systematic, but also made explicit its limits [9].

The research on perceptron - artificial neuron gave promising results, and a lively debate between the two approaches, the neural, connectionist, and the algorithmic, symbolic.

A recent historical review of the research on artificial intelligence can be found in [10], while a good account of the developments of AI, and most of its sub-fields can be found in the review book by Russel et al. [11].

A. Machine Learning

Machine Learning (ML) is a more precisely defined and very active field of research. Wether ML is the research that will realize the paradigm of *strong AI* or not, it is anyways a well defined scientific field of research, with its formalized definitions, and a very active community of scientists developing more and more solutions and achievements.

The goal of ML is to create a system or device which is capable of developing new abilities, not by means of preprogrammed instructions for specific tasks, but rather extracting that from sets of data presented to it. In other words, the key idea of ML is to extract information from data input, and to be able to make predictions about future data [12].

The three main paradigms used in ML are: *supervised learning*, *unsupervised learning* and *reinforcement learning*. In supervised learning, the system is fed with some good-quality input data, along with the correct, desired output. With the right feedback mechanism the goal is to learn from those *training* examples, so that useful and new predictions can be extrapolated, for new data not yet presented.

In the unsupervised learning, the system is presented only with input data, without information about the correct output. The most meaningful application of this model is where an obvious structure is present in the data, e.g. some grouping, and the goal is to find this meaningful structure.

In the reinforcement learning, the system acts as an agent interacting with the environment and learning what are the behaviours that generate rewards.

B. The recognition/classification problems

Among the several fields of application of ML, we will focus on the *recognition problems*, also called *classification problems*, where the highest potential of this discipline has been shown.

This type of problems consist in processing a set of input elements, each belonging to one out of a list of groups or categories, with the task to correctly recognize the category of each element.

C. Deep Learning

Let's observe that to simulate the XOR gate a single neuron is not complex enough, and we have accomplished that more complex task by creating a more complex network, with extra layers.

This suggests that the more difficult and meaningful applications will require more and more complex networks. As the size of the network, and the number of parameters to be chosen grow, the choice of their optimal values can not be done in an inductive and direct way, and a different approach must be found.

II. ARTIFICIAL NEURAL NETWORKS

The Artificial Neural Networks (ANN) represent one of the possible models used to engineer Machine Learning. It is not the only one, and to give few examples of other models we can mention: *Decision trees*, *Bayesian networks*, *Genetic algorithms* and *Support vector machines*.

The ANN is an *abstract model*, and in theory it can be realized and implemented in different ways, using e.g. hardware electronic components, software components, and in principle even mechanical components or other technological implementations. The most common way to implement ANNs is using computer code.

A good account of the developments of ANN can be found in the review book by Russel et al. [11], or in the monograph by Fausett [13] or that by Gupta et al. [14], just to mention a few.

In designing this model, only some aspects of the biological neural system are used, and the precise replication of all the biological characteristics is not the most important goal. To draw an analogy, in designing flying machines humans have definitely taken some fundamental ideas from the observation of birds; however, in pursuing the goal of an efficient flying device, some important differences and deviations from the biological design (flapping wings) have been of fundamental importance.

A. The perceptron - the neuron

The central concept of ANN is the artificial neuron. This can be traced back to the concept of *perceptron*, as introduced in the early studies on AI and ML [5, 15].

The perceptron wants to model a biological neuron, and in the following we will use the term *artificial neuron*, or just *neuron*. An artificial neuron is an object that can accept several *quantitative inputs* $\{x_j\}$; to each input the neuron applies a *weighting factor*, and then it computes the sum of the inputs (weighted sum) $\sum_j w_j x_j$, it adds a *bias* b and then, depending on the sign of this expression the perceptron will have an output or not :

$$output = \begin{cases} 0 & \text{if } \sum_j x_j w_j + b \leq 0 \\ 1 & \text{if } \sum_j x_j w_j + b > 0 \ . \end{cases} \tag{1}$$

This behaviour of an "all or nothing" output of a neuron is based on what it is observed in biological neurons, which are either inactive, or "fire" an output electric signal.

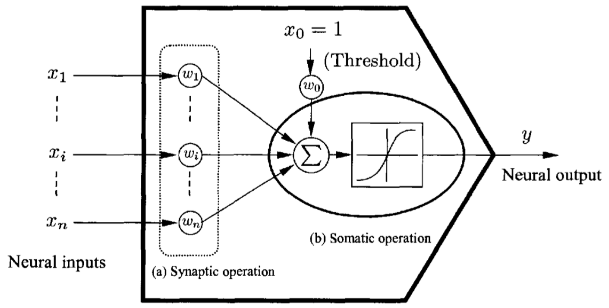
If the neuron is defined like this, it has a linear response with respect to the inputs. It is much more useful to have neurons with a *nonlinear* behaviour, and to obtain this we need to feed the quantity $z = \sum_j x_j w_j + b$ to a nonlinear function $f(z)$, called *activation function*:

$$output = \begin{cases} 0 & \text{if } f(z) \leq 0 \\ 1 & \text{if } f(z) > 0 \ . \end{cases} \tag{2}$$

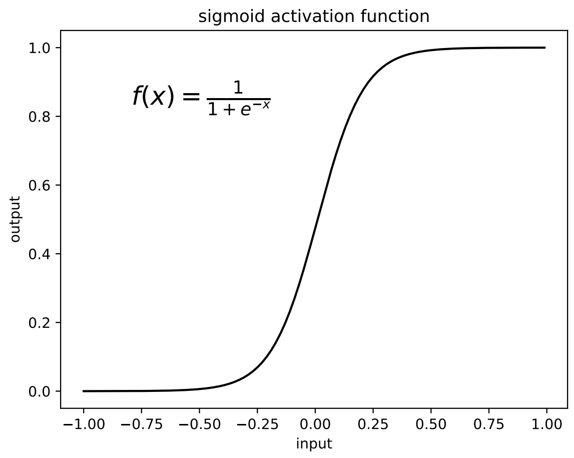
Here we have started to use the formalism of linear algebra, defining the input vector $\vec{x} = \{x_j\}$ and the weights vector $\vec{w} = \{x_j\}$, and using the scalar product $\vec{x} \cdot \vec{w} = \sum_j w_j x_j$. The most used activation function is the *sigmoid*, that is slowly changing for high or small input values but very steep for the mid input values:

$$\sigma(z) \equiv \frac{1}{1 + e^{-z}} \tag{3}$$

which is plotted in fig. 1(b).



(a) Schematics of a single neuron, where the main elements are reported: the multiple inputs, the weights, the activation function, the threshold. Image taken from [14].



(b) Sigmoid activation function. The main characteristics of this activation function are the tails on the left and on the right, and the central slope: this shape provides for the non-linearity. For the sigmoid the range of output values goes from 0 to 1.

Figure 1. The artificial neuron

B. The XOR gate and the network

The research on perceptron - artificial neuron gave promising results, even using just a single neuron, or with several neurons connected in a network, *Artificial Neuron Network*.

The network is created by some *directed connections* i.e. connection with a defined direction (from *transmitting* toward *receiving* neuron(s)), so that the output of a neuron becomes one of the inputs of another neuron.

Since the algorithmic model and the neural model have the same goal of computation, it is reasonable that one model should "simulate" elements of the other model.

A logic gate is an abstract object with some logical input states and some logical output states, which is one of the building blocks of the symbolic-algorithmic approach to computation.

In particular, an *XOR logic gate* (exclusive-OR) is a gate with two logic inputs and one logic output, where the output is *true* only if one input *or* the other is true, but only in this case: if both the inputs are true, the output will be false (i.e. this input is *excluded*). This can be described by the *truth table* (the table with all the possible inputs and corresponding outputs) reported in table I.

It is possible to prove that a single perceptron is unable to implement the behaviour of the *"XOR" logic gate.*

Table I. Truth table of XOR logic gate: the output is *true* only if one of the two inputs (but not both) are true.

input X_1	input X_2	output
false	false	false
true	false	true
false	true	true
true	true	false

It can be proven that to implement this behaviour with a neural network the only way is to use an *hidden layer* i.e. a group of neurons that are connected only with other neurons, and are not directly connected with either the

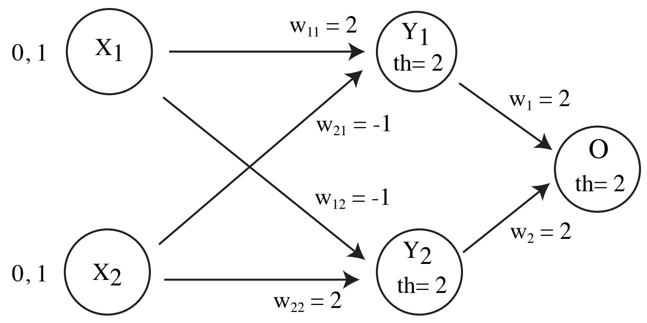


Figure 2. Schematic of a possible implementation of the XOR logic gate, using an ANN with one hidden layer. Details of the numerical values are discussed in the text (see also [13])

inputs of the whole network, or its output.

In fig. 2 we have an example of an ANN that implements the XOR logic gate, which follows one of the early and simpler approaches [5]. In this implementation the logic values of "true" and "false" have been mapped as $false = 0$ and $true = 1$.

The weights are shown in fig. 2 on the links between neurons. This set of values for the weights may have been computed via the training process described above. The thresholds for the two neurons of the hidden layer, and the output neuron are written inside the neurons, and are all set as $t = 2$. Finally, the *activation function* chosen for all the neurons in the identity function. This choice for the activation function means that the weighted sum value is directly passed to the output, if it is above the threshold. In fig. 2, below the input neurons we have also written the possible inputs.

If we manually perform all the computations, compute the input values for the intermediate neurons and the output, for all the possible inputs, we can check that indeed this ANN implements the behaviour of an XOR logic gate.

III. OPTICAL CHARACTERS RECOGNITION (OCR)

A more realistic example of a classification problem (see section I B) is that of Optical Characters Recognition (OCR).

In this task the input is a set of images of handwritten or typed letters. To describe in details this problem, and how to implement a neural network to solve it, let's consider digital images as grids of greyscale pixels, and let's consider to fix the size and resolution of the images. Therefore, the input is a vector of the greyscale values of all the pixels of the input image, and in our implementation we will map each pixel to one input neuron, encoding the pixel greyscale value as a numerical value from 0 (white) to 1 (black).

In fig. 3 we have an example of such image.

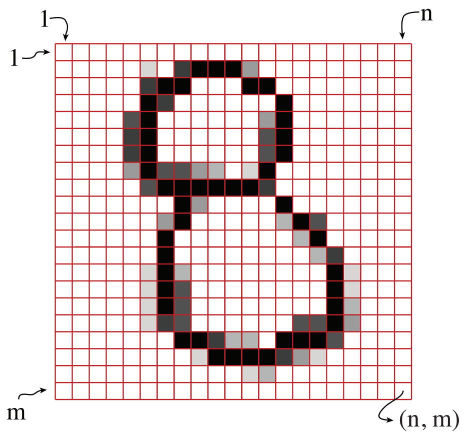


Figure 3. Here is an example of a handwritten character, in particular this is the digit "8". We can number each element of the grid (pixel) with an index, and then a numerical value will express the grey level if the pixel, from white to black.

The task for our ANN is to correctly recognize for each image the corresponding letter or numerical digit that it represents. So, for the OCR the output layer will consist of a number of output neurons equal to the number of symbols that we want to recognize. If we want to recognize only numerical digits we will have 10 symbols, whereas if we want to recognize alphanumeric characters we will have $26 + 10 = 36$ possible outputs, or $(26 * 2) + 10 = 62$ if we want to consider uppercase and lowercase letters.

In fig. 4 we report a summarized sketch of this ANN.

IV. THE TRAINING

The most interesting feature of neural computing is the ability it has to *learn*. In the *supervised training model* this is done by presenting the system with several inputs, for which the correct output is known, and then iterating several cycles of input $\rightarrow$ adjustment of the parameters (weights, bias, threshold) $\rightarrow$ measurement of some *quantitative distance* between the obtained output and the correct output. A method to understand how to change the parameters in an efficient way, in order to minimize the quantitative distance, on the whole set of training inputs is then needed. Once the system is trained well enough, the parameters are "frozen", and the system will be used to respond and give outputs for new, unknown inputs.

For relatively small ANNs the choice of its parameters (*weights*, *biases*, and *thresholds*) can also be done using some ad hoc criteria and some thinking, as we have seen in the XOR gate simulation.

But in more complex (and powerful) ANNs (as seen for the OCR) the number of connections, and therefore the number of weights and biases grows exponentially with

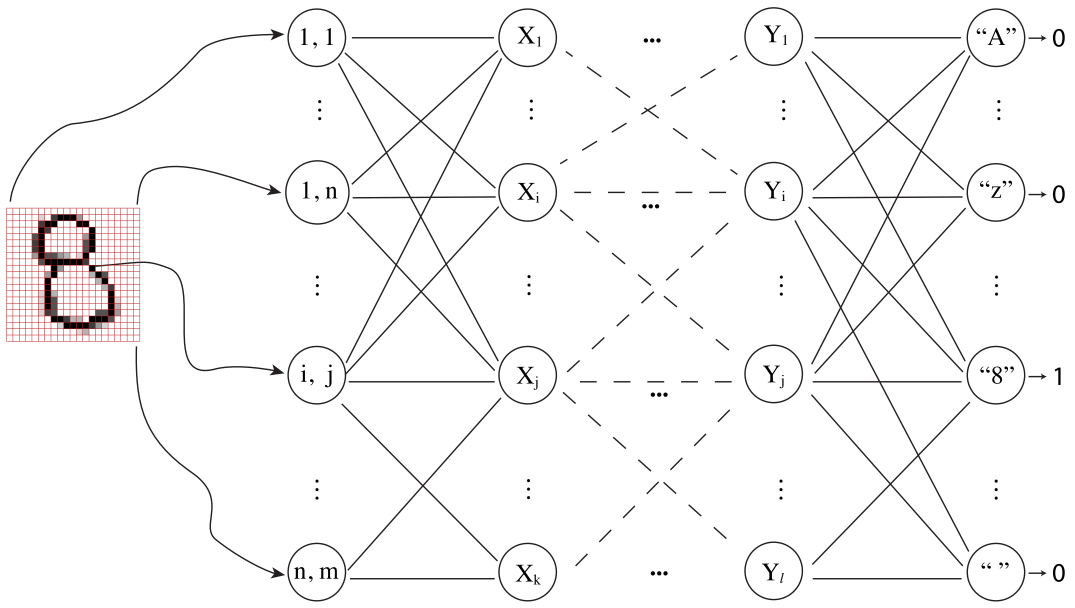


Figure 4. This can be a schematic example of an ANN designed to perform the *optical character recognition* (OCR) task. In input we have the (n, m) pixels of the character figure, as discussed in the caption of fig. 3. In this figure the number of hidden layers is undetermined. As discussed in section I C the number of hidden layers depends on the approach chosen. The output neurons are as many as the possible characters (e.g. the 26 × 2 letters of the english alphabet, uppercase and lowercase, plus the 10 numeric digits). On the right of the image we have shown the output values in an ideal case, where all the output neurons have an output of 0, except for one, that has the output value of 1. This is an ideal case. A more realistic, but still "good" output is one where all the output neurons have a very small numerical output, except for one that has an high output value.

the size (depth), so that a more efficient and systematic approach is needed.

A. The Cost Function

A key element for the training process is the *cost function*, also known as *loss function*. This is a function of all the weights from each connection of the network, and all the biases from each neuron.

We want to use the formalism of *linear algebra* to work on this function, in this and the following sections.

To describe the details of the training process, and in the following sections, we will use the OCR as a working example.

Expanding on the formalism introduced for the single neurons in eq. (2) we will represent an input as a vector, let's call it x, of dimension n, equal to the total number of pixel in a character image.

The *correct output* will also be represented by a vector, let's call it y, with a number of elements m, which is the number of symbols we are going to recognize. The *values* of the elements of the *correct output* vector will be all zero, except for the one corresponding to the correct symbol, which will be one.

Then, also the network parameters (weights and biases) will be represented in vectors: $\vec{w}$ and $\vec{b}$. The number of elements in those vectors will depend on the topology of the network, i.e. the total number of connections, and the total number of neurons respectively.

Once defined all the vectors, we can define a convenient *cost function*, the function that quantitatively measures how "wrong" is a certain choice of the parameters, assessing how different are the obtained outputs, also averaging over all the inputs in the training set:

$$C(\vec{w}, \vec{b}) \equiv \frac{1}{2n} \sum_x |\vec{y}(\vec{x}) - \vec{a}(\vec{w}, \vec{b})|^2 \tag{4}$$

where $\vec{y}$ is the vector of the output values, which is a function of all the network parameters $\vec{w}$ and $\vec{b}$, and has the same dimension (number of elements) of the vector of correct outputs $\vec{y}$, which depends only on the inputs vector.

$$|y - a|^2 = \sum_j (y_j - a_j)^2 \tag{5}$$

[16]

B. Minimization and gradient descent

Once we have set the notation, we can face the task of finding the best choice of weights and biases with a more systematic and efficient approach.

We can use mathematics, in particular calculus, and see the problem as a minimization problem (also known as *optimization problem*): we have to find the minimum of a function of multiple variables.

However, here we face a big “bottleneck” problem, a problem that maybe was the reason for the long period in which the ANN have been neglected by the best part of computer science research.

The problem is that the number of weights and biases of a network of meaningful dimensions grows exponentially, so much so that the problem of minimization becomes intractable with analytical mathematical methods.

In this section we will see a first standard approach to the mathematical problem.

In the later sections we will see more recent methods and techniques, that are more and more efficient, and allow for a speedup of the minimization process.

Those speedups are not just improvements, but a game-changer, which made possible to use ANN for real-life problems, ANNs which have a high number of layers, and therefore a very high number of weights and biases. Making possible to train such deep ANNs brought this research field back to life, and gave it the popularity that it has nowadays.

1. *Gradient descent*

The general technique to use for the minimization of a multiple variables function is the *gradient descent*. It is a standard minimization procedure which starts choosing an initial set of values for the variables (all the weights and biases), and then the gradient of the function is used, to find the “direction”, in the space of the variables along which there is the strongest change in the function. The gradient in turn consists of all the partial derivatives of the function with respect to all the free parameters: given a function $f(\nu_1, \ldots, \nu_n)$ of n variables, the gradient of the function is a n-dimensional vector defined as

$$\nabla f \equiv \left(\frac{\partial f}{\partial \nu_1}, \ldots, \frac{\partial f}{\partial \nu_n} \right) \tag{6}$$

with the property that the scalar product of this vector times any vector $\vec{v}$ in the n-dimensional vector space of the $f(\nu_1, \ldots, \nu_n)$ variables, gives the *directional derivative* of the function along that direction:

$$\nabla f(\vec{\nu}^*) \cdot \vec{v} = D_{\vec{v}} f(\vec{\nu}^*) \tag{7}$$

where $\vec{\nu}^*$ is a fixed set of values for the parameters $(\nu_1^*, \ldots, \nu_n^*)$ and $\vec{v}$ $(v_1, \ldots, v_n)$ is a direction in the same space.

So, the gradient is used in the gradient descent procedure as a tool to find the direction with the most rapid decrease of the value of the cost function, seen mathematically as a multivariable function.

It is useful to give an intuitive and visual description of this procedure.

Imagine the function is a function of only two variables, represented on a (x, y) cartesian plane, and the value of the function is represented on the third z axis (see fig. 5). Choosing a value for the parameters corresponds to a choice of a starting point on the plane. Then, the gradient is a mathematical tool that expresses the *amount of change* (the first derivative) of the function in a given direction. So, if we compute the partial derivatives, and with them the gradient, and once we fix a position in the plane, and a direction, we know how much the function is changing in that direction. If we look at fig. 5, we can imagine to fix a position, as a starting position, then compute the gradient, and look for the direction toward which the cost function is decreasing the most (minimum negative gradient). We will then move of a certain small amount the parameters along that direction, and repeat the procedure. In this way, we should follow a path of descent, toward a minimum of the function.

Remember that the cost function is an average over all the inputs in the training set.

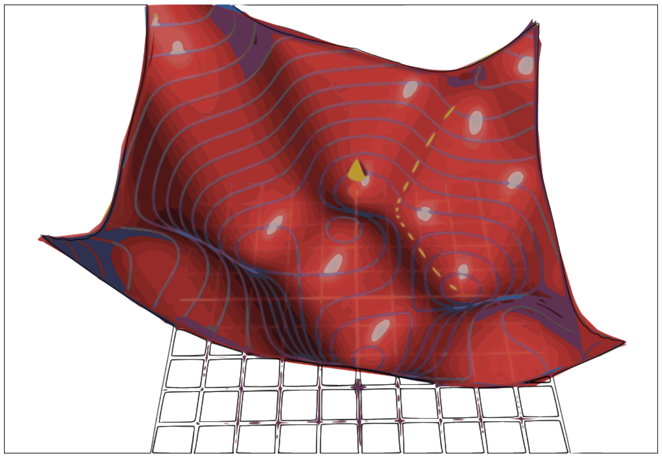


Figure 5. The gradient descent visualized for the two dimensional case. The yellow dashed line represents a path that reaches the minimum.

V. BACKPROPAGATION

Since the power of an ANN depends on the number of neurons, of hidden layers, and of connections, the dimension of the *parameters space* of a powerful ANN may become so big that the procedure described for the gradient descent in the previous section, and in particular the computation of the gradient of the cost function, with all the partial derivatives with respect to all the weights and biases, is not computable, following the definitions of computability elaborated by complexity theory [17, 18].

The ability to find an efficient way to compute the gradient, and in general to minimize the cost function, may represent not just an improvement, but it can make the difference between doable and not doable.

Indeed, one of the breakthroughs of ANN research has been such a technique, called *backpropagation*, which is an efficient and systematic method to compute the gradient of the cost function.

The backpropagation has been devised, in slightly different forms, since the '70s, but its power has been underestimated until the mid '80s.

A. Details of backpropagation

The goal of backpropagation is to have a more efficient way to compute the gradient of the cost function in eq. (4), and in particular to compute all its partial derivatives, with respect to all the weights and biases.

In essence, the backpropagation consists in "unpacking" the dependence of the cost function on the weights and biases, breaking this dependance "layer-by-layer".

Let's look at fig. 4, and eq. (4) and let's follow the process.

Let's start with a very simple ANN, with few hidden layers, and with just one neuron in each layer (see fig. 6) [19].

Let's also call "activation" the output of each neuron, also the neurons in the hidden layers as per the definition of a perceptron (see section II A):

$$a^{(l)} = \sigma\left(w^{(l)} a^{(l-1)} + b^{(l)}\right) \tag{8}$$

where we have assumed that the activation function is the sigmoid $\sigma(z)$, and we have used a superscript to express the layer the neuron belongs to. In this simplified example, with one neuron per layer, this superscript is enough to identify the weight, the bias, and the activation coming from the previous layer.

Now, let's look at the cost function eq. (4), and let's consider only one input, so to simplify the notation, neglecting the averaging over the training set.

The cost function depends on all the weights and all the biases of the network. However, this dependence is "chained", and it propagates layer after layer. So, in computing the partial derivatives of the cost function, we want to "unpack" this chain, and use the chain rule for composite functions. The cost function depends explicitly only on the correct output and the actual output, which in this simplified example is the activation of the only neuron in the output layer: $C(y, a^{(l)}) = \frac{1}{2}|y - a^{(l)}|^2$. In turn, $a^{(l)}$ depends on the (single) weight, the bias, and *the action of the neuron from the previous layer*, as expressed in eq. (8): $a^{(l)}(z) = a^{(l)}\left(w^{(l)} a^{(l-1)} + b^{(l)}\right)$.

So, when we want to compute the partial derivative $\partial C / \partial w^{(l)}$, the chain rule shows that it is possible to break the computation in terms which are all connected to the same layer:

$$\frac{\partial C}{\partial w^{(l)}} = \frac{\partial z^{(l)}}{\partial w^{(l)}} \frac{\partial a^{(l)}}{\partial z^{(l)}} \frac{\partial C}{\partial a^{(l)}}. \tag{9}$$

Only when we compute the partial derivative $\partial C / \partial a^{(l-1)}$ we will have contributions from the other layers. This approach of computing the derivatives one layer at the time, and moving backwards to the previous layers is the reason for the name "backpropagation".

Once we have discussed this simplified case with one neuron per layer, we move to the more realistic case. In order to discuss this more general case we need to introduce more indices in the notation: the weights are identified not only by the layer superscript, but also by two subscripts, indicating the two neurons are connected by the edge of that weight: $w_{i,j}^{(l)}$, so that for each layer there is a matrix of all the weights between the neurons of that layer and the previous one. For the biases and the activations only one subscript will be needed: $b_k^{(l)}$ and $a_k^{(l)}$. Finally, we need to add a subscript also to the cost function, because in a real case we have a contribution to the cost function due to each input of the training set: $C_m^{(l)}$

We are not giving all the details of the computation here, for the details we reference to the historical paper [20], to the book by Michael Nielsen [16], and also to the very informative videos in [19].

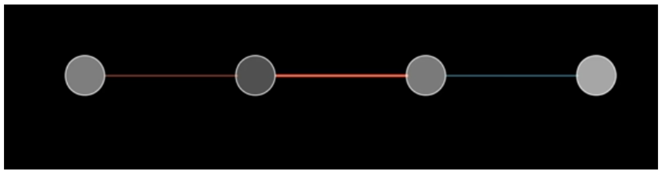


Figure 6. Oversimplified ANN with one neuron per layer.

VI. CONVOLUTIONAL NEURAL NETWORKS

We could say that although the backpropagation made an important breakthrough, optimizing and speeding the training process, and allowing for deep neural network and more meaningful applications, it did not change the Machine Learning paradigm.

The research on ANN has kept progressing, and the more recent results are in the same direction, of making the training process more efficient, allowing for deeper and more complex networks.

In this section we present one of the more recent results, a technique that optimizes the training process.

The main idea behind the Convolutional Neural Networks (CNNs) is that we do not necessarily need to connect each neuron to all the neurons of the previous and the following hidden layers, i.e. we question the efficiency of the *fully connected network* design [21].

Since CNNs have their main application in image processing and image grouping, to better describe the convolutional network design, we will represent the input layer as a 2D grid, so to help the intuition and associate each input neuron to a pixel of the digitized image in input (see fig. 8)

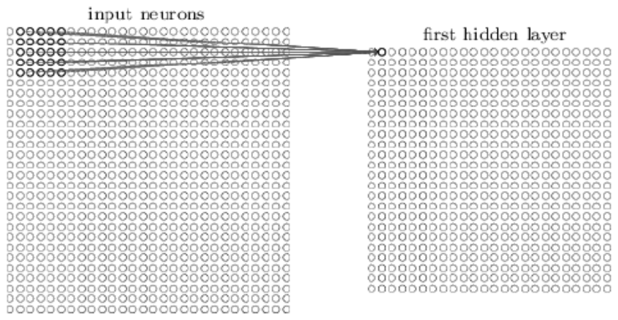


Figure 7. Schematic of the arrangement for a CNN. On the left, the neurons of the input layer, arranged in a 2D grid, mirroring the pixels of the input images. A Local Receptive Field is highlighted on the corner: all its neurons are connected to the same neuron of the first hidden layer, represented on the right.

A. Feature maps and pooling layers

Besides the idea to reduce the number of connections, and abandon the fully connected architecture, there is another main intuition behind the Convolutional design. By connecting all the neurons we not only have a over-abundant number of connections that slow down the training without a comparable advantage. We also have another negative effect: we loose the spatial information about the input, and each input pixel is treated in the exact same way. We may say that in the fully connected architecture we withhold some important information for the training, because we feed all the input at the same way, whereas it is valuable to also give some spatial information, e.g. the fact that some pixels are adjacent, or close, and some other pixels are far apart.

To implement these ideas, in the convolutional model a small region of the input neurons (e.g. a square of 5×5 neurons), called *local receptive field* (LRF), is connected to the same neuron of a first hidden layer.

Then the procedure to create the connections to this hidden layer continues, and the LRF (i.e. its "shape") is shifted of a *step*, 1 or more neurons to one side (stride), and all the neurons in this new LRF are connected to the next neuron in this first hidden layer, and this is repeated until all the input is "covered" and passed to the hidden layer.

The particular detail is that the specific set of weights for the LRF, and the bias of the neuron in the hidden layer are *kept the same*, while the LRF is spanned across the whole input. Since all the weights are fixed, we can think at the hidden layer as a *map*: the weights are chosen so to "respond" to a certain feature, and the neurons in the hidden layer will be activated, or not activated, depending on whether this feature is present in that area or not. A feature can be a certain shape, a certain edge etc. For this reason, this type of hidden layer in the CNN is called a *feature map*: it maps a certain feature in the input.

Together with the feature maps, in the CNN architecture we can use a second type of hidden layer, which are typically connected following a feature map layer, and is called *pooling layer*. A pooling layer can be thought as a summarizing layer, because it takes a certain group of neurons from a feature map layer, let's say a 2×2 square, and it connects them to a single neuron. The criterion can be that of *max-pooling*, where the output of the pooling neuron is the maximum of the outputs of the neurons of the feature map with which it is connected. Another type of pooling is that where a certain average is computed, among the connected neurons of the feature map, e.g. the square root of the sum of the outputs.

The architecture of a CNN does not have a single sequence of layers after layers. It rather has a certain number of feature map layers, all receiving data from the input layer, and each of them possibly passing data to a pooling layer. Then all the pooling layers are connected, usually fully connected, to an output layer (see fig. 8).

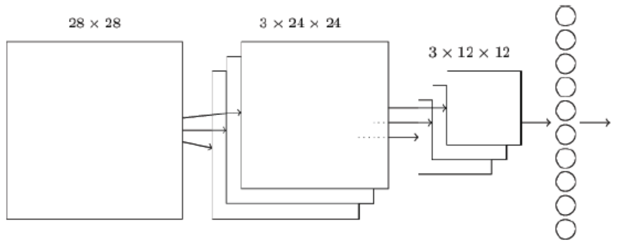


Figure 8. Sketch of a typical implementation of a CNN.

VII. CONCLUSIONS

This introduction to Artificial Neural Networks is rather general, with a summary of the history of the field, and a description of the main elements and the main features of this computational paradigm. We have discussed the example of the OCR in some detail, to see the ANN at work on a concrete task. We have also discussed some more advanced topics: the technique of backpropagation, which allows for a substantial speedup in the training of more deep architectures, and then the more recent idea of Convolutional Neural Networks, which also improve the efficiency.

Although a more detailed description goes beyond the scope of this article, the bibliography can lead the interested scholar to the relevant literature.

[1] John R Searle. Minds, brains, and programs. *Behavioral and brain sciences*, 3(3):417–424, 1980.

[2] John Searle. Minds and brains without programs. *Mindwaves*, pages 209–233, 1987.

[3] A. M. Turing. Computing machinery and intelligence. *Mind*, LIX, 1950.

[4] Roger Penrose. *The Emperor's New Mind.* Oxford University Press, United Kingdom, November, 9 1989.

[5] Warren S. Mc Culloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[6] Hilbert David and Ackermann Wilhelm. Grundzüge der theoretischen logik. 1928.

[7] Alonzo Church. An unsolvable problem of elementary number theory. *American Journal of Mathematics*, 58, 04 1936.

[8] Alan M. Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society*, s2 42, 1937.

[9] Marvin Minsky and Seymour A Papert. *Perceptrons: An introduction to computational geometry.* MIT press, 1969.

[10] Pamela McCorduck. *Machines who think : a personal inquiry into the history and prospects of artificial intelligence.* A K Peters/CRC Press, 2 edition, 2004.

[11] Stuart Russel and Peter Norvig. *Artificial intelligence: A modern approach.* Pearson, 3 edition edition, 2003.

[12] Christopher M. Bishop. *Pattern recognition and machine learning: springer New York.* Springer, 2006.

[13] Laurene Fausett. *Fundamentals of neural networks: architectures, algorithms, and applications.* Prentice-Hall, Inc., 1994.

[14] Madan M Gupta, Liang Jin, and Noriyasu Homma. *Static and Dynamic Neural Networks, From Fundamentals to Advanced Theory. John Walley End Sons.* John Wiley & Sons Inc, 2003.

[15] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65, 1958.

[16] Michael A. Nielsen. *Neural Networks and Deep Learning.* Determination Press, 2015.

[17] Sanjeev Arora and Boaz Barak. *Computational Complexity.* A Modern Approach. Cambridge University Press, 1 edition, 2009.

[18] Stephan Moore and Cristopher Mertens. *The Nature of Computation.* Oxford University Press, USA, 2011.

[19] Grant Sanderson and 3blue1brown. Neural networks. Youtube, August 2018.

[20] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[21] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.