ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/335798012

Learning Deep Learning (CDT-15)

Preprint · September 2019

DOI: 10.13140/RG.2.2.29866.57283

CITATIONS
0

READS
3,699

4 authors, including:

Henrique Ferraz de Arruda
University of São Paulo
48 PUBLICATIONS **212** CITATIONS
SEE PROFILE

Alexandre Benatti
University of São Paulo
10 PUBLICATIONS **3** CITATIONS
SEE PROFILE

Luciano da F. Costa
University of São Paulo
734 PUBLICATIONS **13,325** CITATIONS
SEE PROFILE

Some of the authors of this publication are also working on these related projects:

image analysis View project

Dynamic Systems View project

All content following this page was uploaded by Luciano da F. Costa on 13 September 2019.

The user has requested enhancement of the downloaded file.

Learning Deep Learning (CDT-15)

Henrique F. de Arruda[1], Alexandre Benatti[1], César Henrique Comin[2], and Luciano da F. Costa[1*]

[1]*São Carlos Institute of Physics, University of São Paulo, São Carlos, SP, Brazil*
[2]*Department of Computer Science, Federal University of São Carlos, São Carlos, Brazil*
[*] *luciano@ifsc.usp.br*

September 13, 2019

Abstract

Deep learning consists of using typically large neuronal networks to effectively solve many classification and clustering problems. Its impressive performance derives from using many layers with many neurons, as well as considering large databases. In this text, we briefly describe and discuss in introductory fashion the principles of deep learning as well as some of the principal models. A companion tutorial in Python has been prepared in order to complement the presentation.

"L'apprendimento non esaurisce mai la mente"

Leonardo da Vinci

1 Introduction

In order for humans to interact with their environment, which includes other humans, it is necessary to develop *models* of the entities in the world (e.g. [1]). These models allow not only the *recognition* of important objects/actions, but also provide subsidies for making *predictions* that can have great impact on our lives.

As a consequence of our limited cognitive abilities, the developed models of world entities need to have some level of abstraction, so as to allow a more effective handling and association of concepts, and also as a means to obtain some degree of *generalization* in the representations.

Interestingly, the ability *abstraction* is almost unavoidable for humans, as a consequence of the need to avoid a level of detail that would otherwise not fit our memory and/or processing capacity. So, when we derive a category of a real object such as a pear, we leave out a large amount of detailed information (e.g. color variations, small shape variations, etc.) so as to accommodate the almost unlimited instances of this fruit that can be found.

Provided that we chose an effective set of features to describe the pear, we will be able to recognize almost any instance of this fruit as being a pear, while generally not being able to distinguish between the pears in a tree.

Ideally, it would be interesting that the recognition operated at varying levels of details so that, after recognizing the general type of object, we could process to subsequent levels of increased detail and information, therefore achieving a more powerful performance.

This is the case with several categories that are particularly important to humans, such as faces, plant and animals, as well as actions, among others. In these cases, subcategories are created, leading to increasing levels of information and detail. However, because of limited memory and processing, the problem becomes increasingly *comlex* (e.g. [2]), and we need to stop this sub-categorization at a point that is viable given our needs.

As a consequence of the fundamental importance of pattern recognition for humans, and also of our limitations, interest was progressively invested in developing automated means for performing this ability, leading to areas such as *automated pattern recognition*, *machine learning*, and *computer vision* (e.g. [3]).

Artificial approaches to pattern recognition typically involve two main steps: (a) feature extraction; and (b) classification based on these features (e.g. [4, 1]). Figure 1 illustrates these two stages. While the first stage was initially human-assisted, efforts were focused on the *classification* itself. From the very beginning, neuronal networks were understood to provide a motivation and reference, as a consequence of the impressive power of bi-

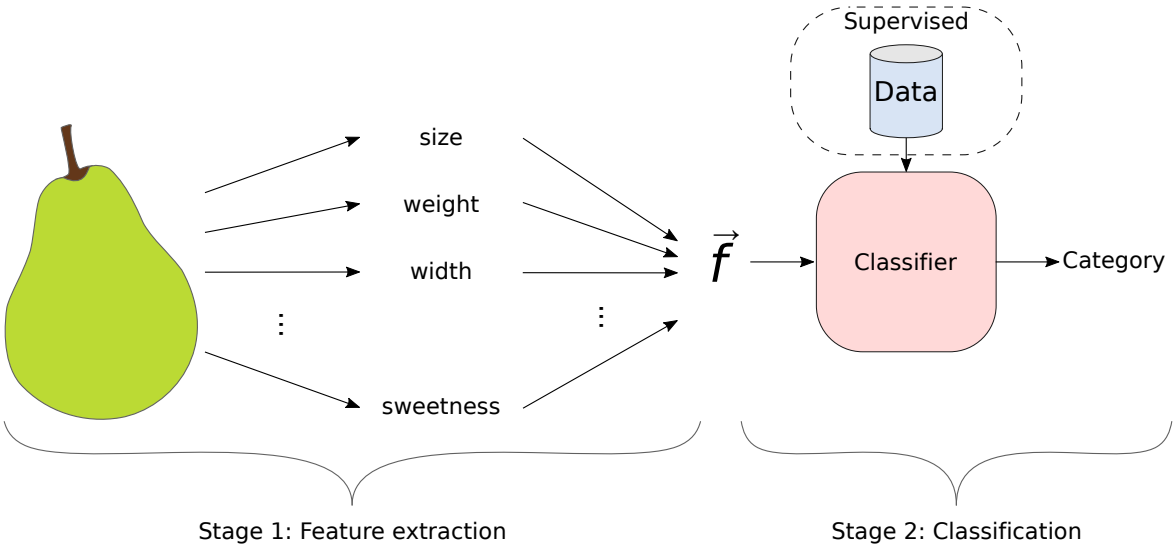

Figure 1: Scheme of classification that starts from the feature extraction step, in which information from the pear is measured, to the classification where the respective category is assigned. In this example, we show a supervised classifier. So, a database (set of training samples) is used.

ological nervous systems (e.g. [5]).

Interestingly, the subject of artificial neuronal networks received successive waves of attention from the scientific-technologic community (for instance, respective to the Perceptron in [6], and Hopfield networks in [7, 8]). These waves, frequently observed in scientific development, are characterized by a surge of development induced by one or more important advances, until further a kind of saturation is reached as a consequence of the necessity of new major conceptual and/or technological progresses. When the situation arises, new waves of development are induced, integrating and complementing the previous developments through new approaches and perspectives.

The great current interest in deep learning represents a major advance, because of the impressive performances that are being obtained. These achievements are a consequence of several interrelated factors. First, we have that computing hardware has developed to unprecedented levels of performance, with emphasis on GPUs (Graphics Processing Units), paving the way to implementing and performing large neuronal algorithms. Thus, many layers are now possible, each incorporating impressive numbers of neurons.

Another important basis of deep learning has been the ever increasing availability of good quality large databases [9]. Added to these, we also have conceptual and methodological advances such as the possibility to automatic feature definition/extraction, as well as the incorporation of novel activation functions [9].

The present text is aimed at providing a brief and relatively accessible conceptual introduction to some of the main aspects and developments underlying deep learning. We also developed a tutorial that comprises examples of all presented deep learning variations, which can be accessed from the following link: `https://github.com/hfarruda/deeplearningtutorial`.

This didactic text is organized as follows. Section 2 describes traditional neuronal networks. In Section 3, we cover the main characteristics of deep learning networks, which include activation functions, computer architectures, as well as some of the main deep learning architectures.

2 Traditional Neuronal Networks

The idea of creating a classifier based on neurons starts on a single unity whose dendrites can receive the input information that is weighted by $\boldsymbol{W}$, the cellular body sums up the data, and the axons give the classification activation, which is controlled by a given function (called activation function). More information regarding the activation functions are shown in Section 3.3. In order to attribute the correct class to a given input data, it is necessary to associate the more appropriate weights, by using a given training method. One possibility is to optimize $\boldsymbol{W}$ according to an error function, where the error is optimized as follows

$$w_i(n) = w_i(n-1) + \alpha\,\kappa\,\epsilon, \tag{1}$$

where $w_i \in \boldsymbol{W}$, α is the learning rate, κ is the input data, and ϵ is the error. Interestingly, this simple methodology can classify data from two classes that are linearly

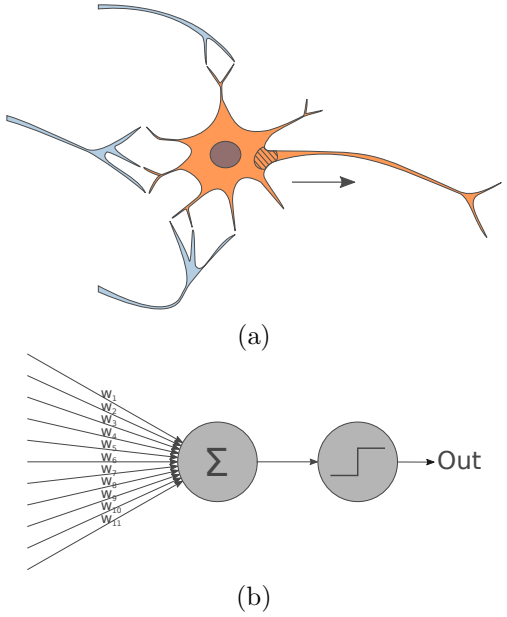

Figure 2: (a) represents a biological neuron, and (b) is its particular model version. The following parts compose a neuron: dendrites, that receives the signal; the cellular body; the implantation cone (represented as the striped region), in which occurs the integration; and the axons that transmit the signal to the next neuron. The data comes from the left side, in which each value is multiplied by the respective w_i weight, all these values are summed up, and an activation function gives the result.

separated. Se an example of neuron in Figure 2.

In order to represent convex regions, sets of neurons have been considered, which are organized as a network [10]. The more straightforward manner is the use of the *Multilayer Perceptron* (MLP) [10]. In this case, the neurons are organized into three types of layer:

- Input layer: the first layer of the network (data input);
- Hidden layer: receives information from a previous and, after the sum and activation, transmit the data to the following layer. In the same network, it is possible to have as much hidden layers as necessary;
- Output layer: gives the classifier answer.

In MLP, all the neurons from a previous layer can be connected to the next, which is called *dense*. The training step consists of the method of back propagation [10] being similar to the approach employed to a single neuron. More specifically, the training data is imputed in the network, and the weights are optimized to decrease the error function from the output to the input layer. Interestingly, it was found that one hidden layer, for a sufficient large number of neurons, is capable of learning any possible shape [11], which is called *universal aproximation theorem*. So, theoretically, this number of layers is enough to any different problem. However, the specialists commonly use at least two hidden layers because it was found to decrease the learning time and improve the accuracy [12].

3 The Deep Learning Framework

One of the essential points of deep-learning is the capability of the network to extract features directly from the data. While feature extraction and classification are performed in standard machine learning methods, in deep-learning the network can learn the features from the raw data. Figure 3 illustrates the similarities and differences between a typical neuronal network and a convolutional deep learning network.

3.1 Optimization

Optimization is one of the key points of deep learning. This step consists of minimizing the loss function during neuronal network training. The loss function, which measures the quality of the neuronal network in modeling the data, is used in order to optimize the network weights ($\boldsymbol{W}$). There are several functions that can be used as loss functions, some example include: *Mean Square Error* (MSE), *Mean Absolute Error* (MAE), *Mean Bias Error* (MBE), *SVM Loss*, and *Cross entropy loss*. The chosen function depends on the deep learning network type and the performed task.

The method used for optimization is called *Optimizer*. This optimization allows the classifier to learn its weights $\boldsymbol{W}$ with respect to the training data. Because it is not possible to know where is the global minimum of this function, several methods have been considered including *gradient descent*, *stochastic gradient descent*, and *Adam* [13] The latter, one of the most often adopted methods, consists of an efficient method for stochastic optimization. As in the stochastic gradient descent, Adam also employs random sub-samples, called minibatches. For each of the optimized parameters, one individual adaptive learning rate is used, and the parameters are estimated from the first and second moments of the gradients. This method is indicated for problems involving a significant number of parameters [13]. Another advantage of Adam is that it requires relatively reduced memory.

3.2 GPUs

The Graphics Processing Unit (GPU) was created to deal with graphical applications, such as games. By considering the high processing power of these GPUs, the manufacturers created a novel type of boards, called General

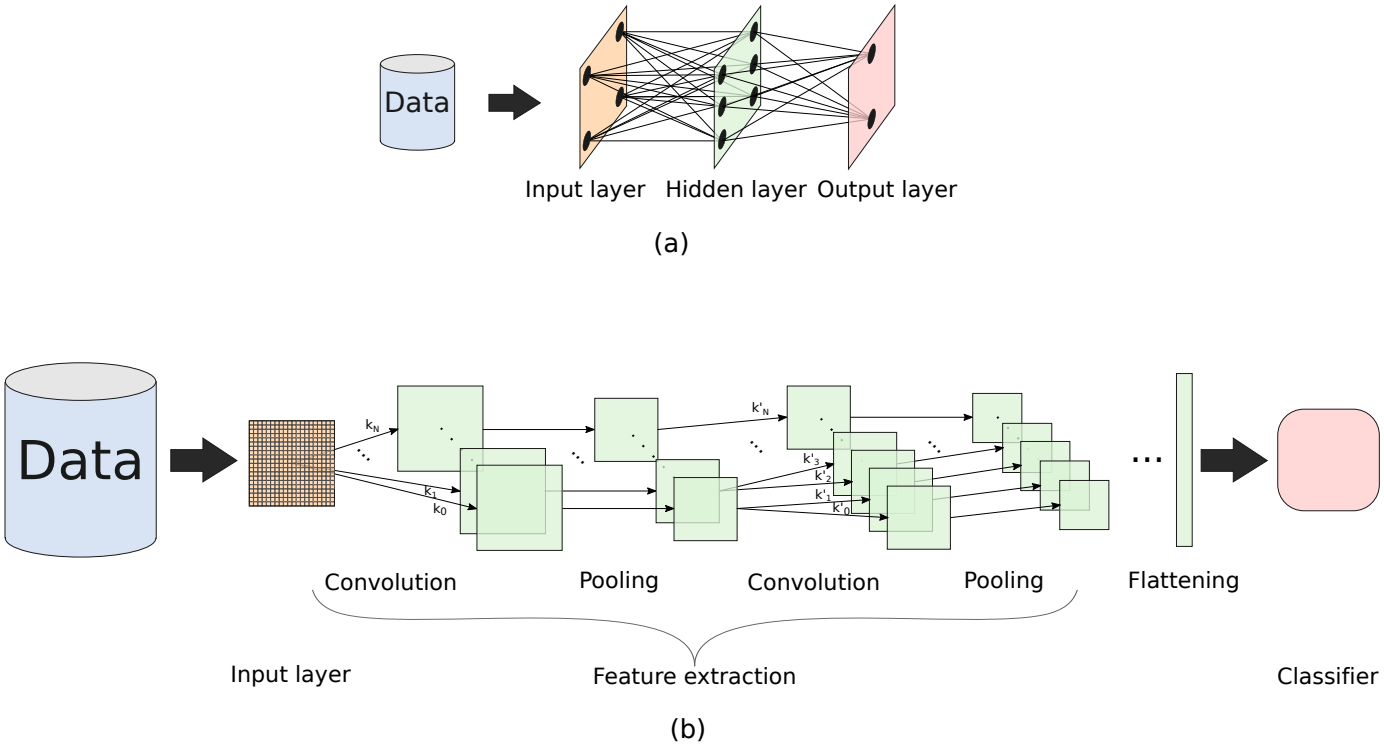

Figure 3: A simple multilayer perceptron (a), and a convolutional deep learning network (b). A convolutional network involves layers dedicated to *convolution*, *pooling*, and *flattening*. Each matrix of the convolution layer is associated with a given kernel k_i. Often a feedforward network is adopted for the *classifier* component.

Purpose Graphics Processing Unit (GPGPU) that can be applied to a wider range of applications. One of the advantages of the GPGPUs, when comparing with GPUs, is that programs of GPGPUs can be implemented in a simpler way. Consequently, many libraries have been developed, which include more efficient methods of linear algebra, computer graphics, image processing, and deep learning.

By comparing Central Processing Units (CPUs) and GPGPUs, typically the GPGPUs have a considerably higher number of cores (see Figure 4). However, only some specific applications can be executed in GPGPU, which are characterized by data parallelism tasks. For instance, the GPGPUs are efficient when a given operation is computed over many elements of an array. Some disadvantages of using GPGPUs include the high cost of the data transfer between the CPU RAM (Random-Access Memory) and the board memory, and the limited memory size, among others. The GPGPUs typically cannot replace CPUs, but many novel approaches have become achievable with this technology, including deep learning.

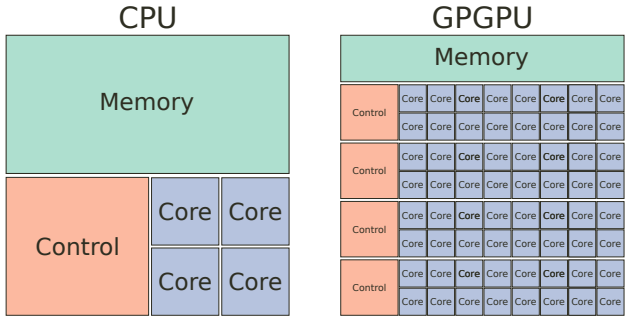

Figure 4: A simplified comparison between CPU and GPGPU. The CPU cores are more powerful and can be employed to execute complex tasks in parallel, while the GPGPU have lots of cores that are more specific to performing massive processing of data.

3.3 Activation Functions

In biological neurons, the cellular body sums up the input stimulus, and the output is controlled by a respective *activation function*. In Figure 5 we show some of the main types of activation functions. In the case of the step function [14], if the integrated stimulus intensity is lower than zero, the neuron is considered *unactivated*, yielding zero

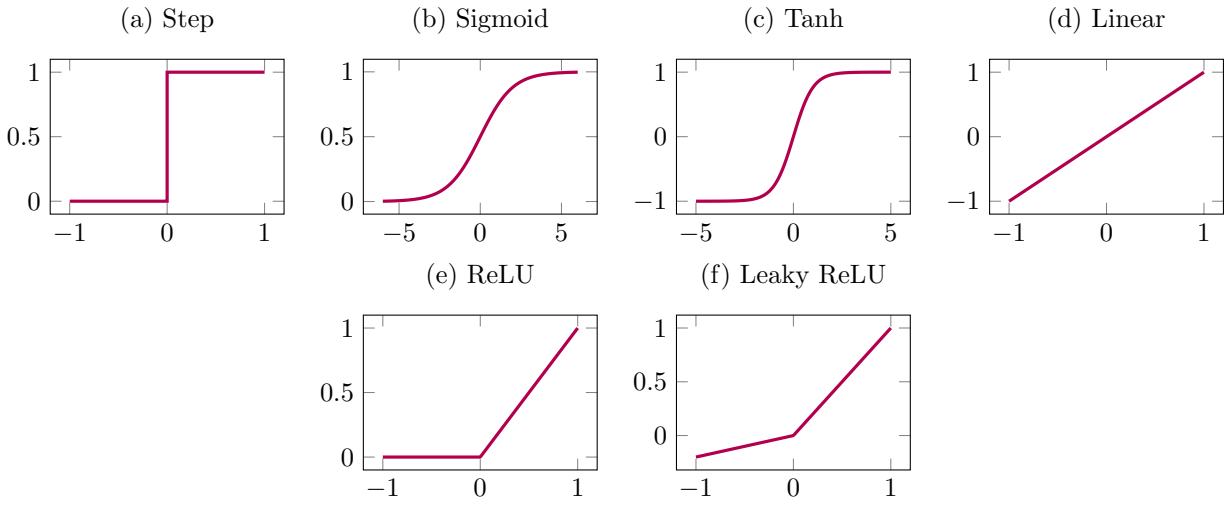

Figure 5: Examples of activation functions, used in specific neuronal networks-based solutions.

as result. Otherwise, the neuron returns one and is considered *activated* (see Figure 5(a)). The step function is typically employed for classification of linearly separable data.

Other activation functions can also be employed for binary (i.e. two classes) classification. For example, we have the *sigmoid* function [9], which can be interpreted as a probability as it returns values between zero and one (shown in Figure 5(b)) this function is defined as

$$f(x) = \frac{1}{1 + e^{-x}}, \tag{2}$$

where x is the value of the cellular body sum. Another alternative is the *hyperbolic tangent* [15], which is defined as

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}. \tag{3}$$

In this case, the function returns positive or negative values when the input is positive or negative, respectively, as shown in Figure 5(c). Due to this characteristic, the hyperbolic tangent is typically employed in tasks involving many negative inputs.

Another possibility is the *identity function* [9], also called *linear function*, in the jargon of the area. In this case, the input and output are exactly the same, as can be observed in Figure 5(d). This function is typically employed for regression tasks. In the case of the convolutional neural network (CNN, see Section 3.5), the most common activation function is the *Rectified Linear Units* (ReLU) [9], which is a function defined as

$$f(x) = \max(0, x). \tag{4}$$

This functions is shown in Figure 5(e). In the case of image data, the value of the pixels are given by positive numbers. So the input layer does not have negative values. This function is known to be easily optimized and preserve properties having good generalization potential.

Alternatively, the Leaky Rectified Linear Units (Leaky ReLU) function can be employed instead of ReLu. The difference is that the Leaky ReLU returns output values different from zero when the inputs are negative. In some situations, the Leaky ReLU was found to reduce the training time. This function is defined as

$$f(x) = \max(\alpha x, x), \tag{5}$$

where α is the parameter that controls the negative part of the function. An example of this function is illustrated in the Figure 5(f).

The *softmax* function [9] can be used to deal with classification problems in which there are many distinct classes. This function is defined as follows

$$f(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}}, \tag{6}$$

where x_i is the output of a neuron, $\boldsymbol{X}$ is the set of outputs of the neurons in output layer, $x_i \in \boldsymbol{X}$, and N is the number of neurons in the output layer. Because the sum of the exponential values normalizes this function, it can be understood as a probability distribution.

3.4 Deep Learning Critical Aspects

This subsection is dedicated to briefly describing the characteristic aspects of deep learning.

3.4.1 Bias

The concept of bias consists of incorporating a fixed value, $\boldsymbol{b}$, as input to the neuronal layer. This value allows the activation function to adapt in order to better fit the data.

Bias is mathematically represented as

$$\boldsymbol{Y} = f(\boldsymbol{X} \cdot \boldsymbol{W} + \boldsymbol{b}), \tag{7}$$

where $f(\cdot)$ is a given activation function and $\boldsymbol{Y}$ is the next layer input data.

3.4.2 One Hot Encoding

One possibility to deal with categorical features (e.g., car brands, nationalities, and fruits) is to map the categories into integer numbers. However, the order of the numbers is arbitrary and can be interpreted by the classifier as a ranking. Another solution consists of assigning a separated variable to each category. An example regarding fruits can be found in Figure 6. This approach is called one hot encoding.

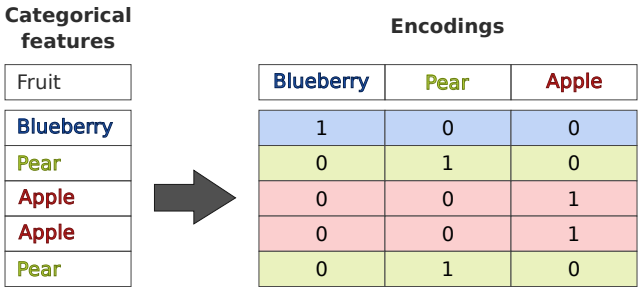

Figure 6: Example of One Hot Encoding, where the fruits are converted into a sparse matrix.

3.4.3 Pooling

This process is used in CNNs, typically after the convolution, for reducing the dimensionality of a given matrix, by first partitioning each matrix in an intermediate layer and then mapping each partition into a single value. There are many possibilities of *pooling*. For example, the *max pooling* selects the maximum value from each window; the *min pooling* considers the minimum value instead, among many other possibilities. See an example in Figure 7.

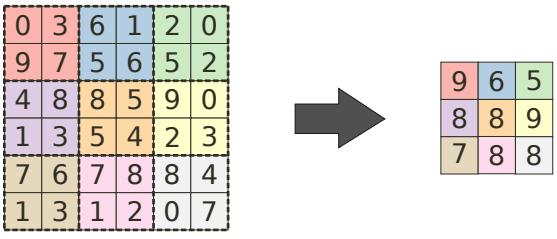

Figure 7: Example of max pooling, in which the highest number of each window is selected and assigned to the new, reduced matrix.

3.4.4 Flattening

This technique is employed in CNNs to convert a 2D matrix (or a set of matrices) into a 1D vector, as

$$\begin{bmatrix} X_{1,1} & X_{1,2} & \cdots & X_{1,M} \\ X_{2,1} & X_{2,2} & \cdots & X_{2,M} \\ \vdots & \vdots & \ddots & \vdots \\ X_{N,1} & X_{N,2} & \cdots & X_{N,M} \end{bmatrix} \xrightarrow{Flattening} \begin{bmatrix} X_{1,1} \\ X_{1,2} \\ \vdots \\ X_{1,M} \\ X_{2,1} \\ X_{2,2} \\ \vdots \\ X_{2,M} \\ \vdots \\ X_{N,M} \end{bmatrix}, \tag{8}$$

where $N \times M$ is the dimension of the input matrix $\boldsymbol{X}$. By considering a matrix set, the resultant vector represents the concatenation of the vectors respectively to all of the matrices.

3.4.5 Overfitting

Overfitting happens when the model fits many details of the training data at the expense of undermining its generality for classifying different data. Figure 8 illustrates an example of this behavior. Some of the possible approaches to address overfitting are discussed in the following.

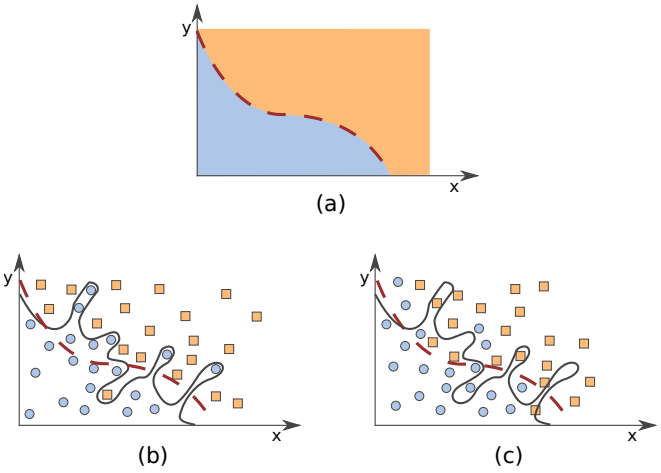

Figure 8: Example of overfitting while classifying samples from two classes, represented by blue circles and yellow squares. The dashed lines indicate the real separation between regions, and the black lines indicate the separation found by a classifier. The actual problems is known to have classification regions as in (a), but different samplings from this problem can lead to rather different classification curves, as illustrated in (b) and (c), in case this curve adheres too much on each of the specific sampled data sets.

3.4.6 Dropout

Dropout was proposed in order to minimize the problem of overfitting. More specifically, the objective of this approach is to avoid excessive detail by replacing a percentage of the correct values of a given layer with zeros. The percentage of zeros is the parameter of this technique. The success of the Dropout derives from the fact that the neurons do not learn every too much detail of the instances of the training set.

3.4.7 Batch Normalization

Batch normalization [16] is based on the idea of normalizing the input of each layer to zero mean and unit standard deviation. The advantages of applying this technique include the reduction of the number of training steps and, consequently, a decrease of the learning time. This effect occurs because it allows the neurons from any layer to learn separately from those in the other layers. Another advantage is that, in some cases, the batch normalization can mitigate the overfitting. In these cases, the use of Dropout can be unnecessary.

3.4.8 Weight Regularization

Typically, overfitting leads to large values for the weights $\boldsymbol{W}$. Thus, overfitting may be reduced by constraining the weights to have small values, that is, by regularizing the values of the weights. This technique consists of adding penalties to the loss function during the optimization step. Some possibilities of weight regularization were proposed [9], such as $L1$, $L2$ (also called weight decay), and $L1L2$. $L1$ and $L2$ are defined as the sum of the absolute weights and the sum of the squared weights, respectively, and $L1L2$ employs the sum of both regularizations, which are defined as

$$L1 = \sum_i |w_i| \; l_1, \tag{9}$$

$$L2 = \sum_i \sqrt{w_i} \; l_2, \tag{10}$$

$$L1L2 = L1 + L2, \tag{11}$$

where $l1$ and $l2$ set the amount of regularization and w_i are elements of the matrix $\boldsymbol{W}$.

3.5 Types of Deep Learning Networks

In order to deal with a variety of problems, many neuronal network topologies have been proposed in the literature [9]. Here, we describe some of the most used deep learning topologies, and comment on their respective applications.

3.5.1 Feedforward

Feedforward is one of the first artificial neural network proposed in the literature [17]. In spite of its simplicity, this network is still used nowadays, including deep learning. In a feedforward structure, the information moves in a single direction (from the input nodes to the output, through the hidden layers), without loops. Figure 9 shows a typical example of feedforward network for binary classification, having a single neuron in the output layer.

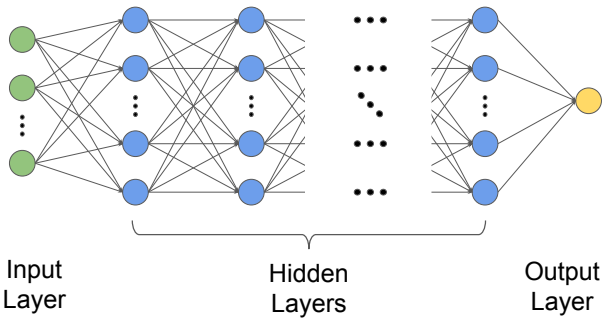

Figure 9: Example of a feedforward network that can be employed in binary classifications.

In Deep Learning Tutorial - 1, two main examples of the use of this network can be found. In the first example, we employ the feedforward network in binary classification, more specifically to distinguish wine from two regions, by using the wine dataset. The second consists of an analysis of the same dataset, which can be classified into three classes, according to their regions. In this case, we show an example of the use of softmax as the activation function. In addition to the classification illustrated task, the feedforward networks are also employed to regression tasks.

3.5.2 Convolutional Neural Network

A convolutional neural network (CNN) [9] is often applied for visual analysis. CNNs can be particularly efficient in tasks such as objects detection, classification, and face detection. This is to a great extent allowed by the fact that these networks can automatically learn effective features. An example of CNN is shown in Figure 3.

The first layer of a CNN is a matrix corresponding to an image. The hidden layers are associated to specific convolutions, followed by respective pooling layers. These two types of layers are repeated many times, alternately. The matrices are then converted into an 1D vector by using the process called flattening. Finally, the vector is sent to a classifier, e.g., a feedforward network. Many variations of this network can be found in the literature. Usually, the

dropout technique does not tend to be particularly effective when applied to CNNs because a very large number of nullifying operations would be need to be applied in order to counteract the large redundancy commonly found in visual data.

In Deep Learning Tutorial - 2, we present a tutorial regarding classification, using the well-known CIFAR10, which consist of a dataset of colored images with 10 classes.

3.5.3 Recurrent Neural Network

Recurrent neuronal networks (RNN) consists of a class of neuronal networks with recurrent layers, as illustrated in Figure 10. In these layers, for each neuron, the output is re-inserted as an input. The hyperbolic tangent is employed as the activation function in hidden layers. This recurrent behavior can be understood as a type of memory, which gives rise to different ways of processing sequences or vectors. This type of network has been employed to solve various problems including speech recognition, text translation, language modeling, and image captioning, among others.

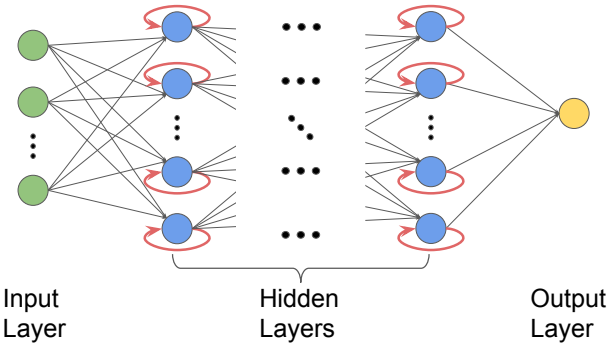

Figure 10: Example of RNN, which is similar to the structure of the feedfoward network, but with recurrent neurons.

One problem that can affect the RNNs is called *vanishing gradient*, which consists in the gradient of loss functions vanishing to zero during the training stage. In order to address this problem, it is possible to use the Long Short-Term Memory (LSTM)[1]. For each LSTM neuron, there is a different set of rules that involves some activation functions (sigmoids and hyperbolic tangents). These functions control the flow of incoming information so as to boost the output, consequently reducing the effect of the vanishing gradient.

In Deep Learning Tutorial - 3, we present a deep LSTM learning model able to predict the Bitcoin prices along time by using the input as a temporal series.

[1]More information about the LSTM can be found in `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`.

3.5.4 Boltzmann Machine

The Boltzmann machine is a stochastic neuronal network where all neurons are non-directly interconnected. The types of neurons used in this model are divided into *visible* and *hidden*, the information being input into the former type of neurons, which can have their values modified. *Simulated annealing* is used in the training step instead of gradient descendent. One disadvantage of this network is its high computational cost due to the high number of connections increasing exponentially with the number of neurons.

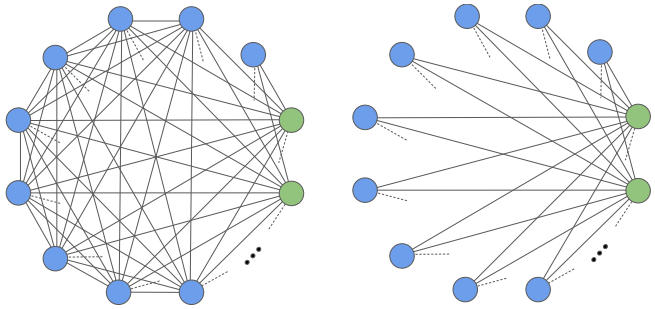

Figure 11: (a) Example of Boltzmann Machine network and (b) example of Restricted Boltzmann Machine network, in which the green and blue nodes represent visible and hidden neurons, respectively.

One possible solution to the problem of high computational cost is the restricted Boltzmann machine (RBM) [18], which is a variant of the Boltzmann machine. In this case, the nodes are divided into two layers, that represent visible and hidden neurons. The neurons from one layer are connected with all the neurons of the other layer. Furthermore, there are no connections between nodes in the same layer. Another difference is the employed training step of RBM, which can be the contrastive divergence (CD) algorithm [18]. Among the many possible applications, we can list dimensionality reduction, classification, collaborative filtering, feature learning, topic modeling, many-body quantum mechanics, and recommendation systems.

In Deep Learning Tutorial - 4, we provide an example of RBM for a recommendation system of CDs and Vinyls.

3.5.5 Autoencoders

Autoencoders consist of a deep learning model that generates a coding representation from a given data. One example of autoencoder is shown in Figure 12. The first part of the network is used to create the code (encoder), and the second is responsible for recovering the original data (decoder). The quality of training is measured by considering the differences between the input layer and

the output layer. After the training step, the decoder and the output layer are removed, and the code layer becomes the output of the network. There are many applications of autoencoders, which include dimensionality reduction, information retrieval, as well as several computer vision tasks. In the latter, the encoder and decoder layers are typically convolutional.

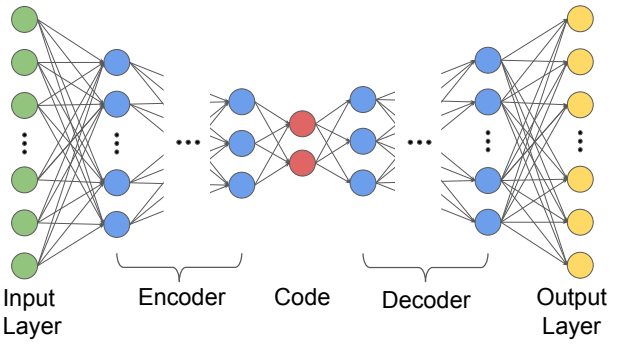

Figure 12: Example of autoencoder network. After the training step, the code layer becomes the output of the network.

In order to illustrate an autoencoder application, we use the *Fashion MNIST* dataset, which comprises some different types of clothes. By using the resulting codes, we project the data using a Principal component analysis (PCA), available in Deep Learning Tutorial - 5.

3.5.6 Generative Adversarial Networks

Generative Adversarial Networks (GANs) are supervised deep learning models capable of generating, through learning, patterns from data and noise. These networks consist of two parts, the *generator*, and the *discriminator* (see Figure 13). For instance, for a GAN that creates characters, the generator is responsible for creating the desired character, and the discriminator monitors the quality of the generated character. The training step is repeated many times, and the discriminator is progressively made more strict regarding the training dataset. Applications adopting GANs include the generation of images from texts, videos from images, and text from videos, among others. Also, GANs are known to be sensitive to the variation of the network parameters [19, 20].

In Deep Learning Tutorial - 6, we present an example regarding handwritten character generation, using the MNIST (Modified National Institute of Standards and Technology) dataset to train a GAN. As a result, we create a network that automatically generates handwritten characters.

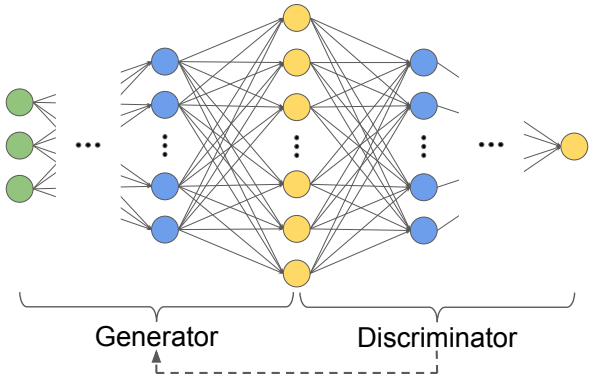

Figure 13: Example of Adversarial networks, which are divided into two parts, the generator and discriminator.

4 Performance Evaluation

Performance evaluation methods can be applied to each deep learning configuration, and are often decisive for enhancing performance and better understanding the obtained results given the adopted configurations. In the case of supervised methods, one of the most common evaluation approach is the k-fold. This approach consists of partitioning the data into k groups with the same size. For each step, a group is selected, as the test group, while the remainder are considered as the training group. This process is repeated k times for all possible test groups, and then some accuracy index is computed. The average and standard deviation of this accuracy are calculated. A high accuracy average means that the method reached a high performance and the high values of standard deviation suggest that probably the classifier is overfitted (ex [21]).

For the cases in which a massive combination of parameters needs to be tested, another approach can be employed to avoid overfitting. In this case, the data is divided into three sets, namely train, validation and test. The training set is used for optimizing the model weights. The validation set is used for estimating the model performance and for tuning hyperparameters (e.g., number of layers, dropout rate, number of training epochs). After a final model is obtained, the test set is used for estimating the model performance on previously unseen data.

5 Conclusion

Deep learning is a relatively recent neuronal network-based approach that has been used to effectively solve many problems involving classification and clustering. As such, these networks are progressively being incorporated into the most diverse applications, ranging from auto-

Models	Learning	Main Applications	Information Flow	Links to Tutorials
Feedforwrd	Supervised	Classification and Regression.	Single Direction	Tutorial - 1
CNN	Supervised	Computer Vision.	Single Direction	Tutorial - 2
RNN	Supervised	Temporal Series.	With Loops	Tutorial - 3
RBM*	Unsupervised	Computer Vision, Recommender Systems, Information Retrieval, and Data compression, etc.	Undirected	Tutorial - 4
Autoencoder	Unsupervised	Information Retrieval and Data compression.	Single Direction	Tutorial - 5
GAN	Semi-supervised	Generation of Images, Audio Synthesis, NLP†, and Temporal Series.	Single Direction	Tutorial - 6

Full tutorial available at `https://github.com/hfarruda/deeplearningtutorial`

Table 1: Comparison among the models considered in this didatic text. *RBMs are normally employed as a part of the *deep belief networks*. †NLP means Natural Language Processing. The last column presents links to tutorial elaborated for each model.

mated movies subtitles to self-driving cars.

In principle, deep learning structures consist of large neuronal networks involving many neurons and considering very large datasets, as well as GPGPUs. In addition, some new concepts and methods have been incorporated in this approach, including autoencoding, diverse activation functions, as well as overfitting prevention.

In the present work, we provided a brief introduction to the main elements underlying deep learning, including its motivation, basic concepts, and some of the main models. Table 1 summarizes the revised models and some of their respective characteristics. A tutorial in Python has been specially prepared to serve as a companion to this work, illustrating and complementing the covered material (`https://github.com/hfarruda/deeplearningtutorial`).

It is hoped that the reader will be motivated to probe further into the ever expanding related literature.

Costa's Didactic Texts – CDTs

CDTs intend to be a halfway point between a formal scientic article and a dissemination text in the sense that they: (i) explain and illustrate concepts in a more informal, graphical and accessible way than the typical scientic article; and(ii) provide more in-depth mathematical developments than a more traditional dissemination work.

It is hoped that CDTs can also integrate new insights and analogies concerning the reported concepts and methods. We hope these characteristics will contribute to making CDTs interesting both to beginners as well as to more senior researchers.

Though CDTs are intended primarily for those who have some experience in the covered concepts, they can also be useful as summary of main topics and concepts to be learnt by other readers interested in the respective CDT theme.

Each CDT focuses on a few interrelated concepts. Though attempting to be relatively self-contained, CDTs also aim at being relatively short. Links to related material are provided in order to complement the covered subjects.

The complete set of CDTs can be found at: `https://www.researchgate.net/project/Costas-Didactic-Texts-CDTs`.

Acknowledgements

Henrique F. de Arruda acknowledges FAPESP for sponsorship (grant no. 2018/10489-0). Alexandre Benatti thanks Coordenao de Aperfeioamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001. Luciano da F. Costa thanks CNPq (grant no. 307085/2018-0) and NAP-PRP-USP for sponsorship. César H. Comin thanks FAPESP (Grant Nos. 15/18942-8 and 18/09125-4) for financial support. This work has been supported also by FAPESP grants 11/50761-2 and 2015/22308-2.

References

[1] L. da F. Costa, "Modeling: The human approach to science (cdt-8)," 2019.

[2] L. da F. Costa, "Quantifying complexity (cdt-6)," 2019.

[3] L. da F. Costa and R. M. Cesar Jr, *Shape analysis and classification: theory and practice*. CRC Press, Inc., 2000.

[4] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern classification*. John Wiley & Sons, 2012.

[5] F. R. Monte Ferreira, M. I. Nogueira, and J. DeFelipe, "The influence of james and darwin on cajal and his research into the neuron theory and evolution of the nervous system," *Frontiers in neuroanatomy*, vol. 8, p. 1, 2014.

[6] I. Stephen, "Perceptron-based learning algorithms," *IEEE Transactions on neural networks*, vol. 50, no. 2, p. 179, 1990.

[7] J. D. Keeler, "Comparison between kanerva's sdm and hopfield-type neural networks," *Cognitive Science*, vol. 12, no. 3, pp. 299–329, 1988.

[8] B. Xu, X. Liu, and X. Liao, "Global asymptotic stability of high-order hopfield type neural networks with time delays," *Computers & Mathematics with Applications*, vol. 45, no. 10-11, pp. 1729–1737, 2003.

[9] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[10] S. S. Haykin *et al.*, *Neural networks and learning machines/Simon Haykin*. New York: Prentice Hall,, 2009.

[11] R. Hecht-Nielsen, "Kolmogorovs mapping neural network existence theorem," in *Proceedings of the international conference on Neural Networks*, vol. 3, pp. 11–14, IEEE Press New York, 1987.

[12] M. M. Poulton, "Multi-layer perceptrons and backpropagation learning," in *Handbook of Geophysical Exploration: Seismic Exploration*, vol. 30, pp. 27–53, Elsevier, 2001.

[13] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[14] A. K. Jain, J. Mao, and K. Mohiuddin, "Artificial neural networks: A tutorial," *Computer*, no. 3, pp. 31–44, 1996.

[15] P. Sibi, S. A. Jones, and P. Siddarth, "Analysis of different activation functions using back propagation neural networks," *Journal of Theoretical and Applied Information Technology*, vol. 47, no. 3, pp. 1264–1268, 2013.

[16] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[17] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural networks*, vol. 61, pp. 85–117, 2015.

[18] G. E. Hinton, "Training products of experts by minimizing contrastive divergence," *Neural computation*, vol. 14, no. 8, pp. 1771–1800, 2002.

[19] K. Roth, A. Lucchi, S. Nowozin, and T. Hofmann, "Stabilizing training of generative adversarial networks through regularization," in *Advances in neural information processing systems*, pp. 2018–2028, 2017.

[20] L. Metz, B. Poole, D. Pfau, and J. Sohl-Dickstein, "Unrolled generative adversarial networks," 2016.

[21] G. C. Cawley and N. L. Talbot, "On over-fitting in model selection and subsequent selection bias in performance evaluation," *Journal of Machine Learning Research*, vol. 11, no. Jul, pp. 2079–2107, 2010.

View publication stats