Revista Brasileira de Ensino de Física, vol. 44, e20220101 (2022)
www.scielo.br/rbef
DOI: https://doi.org/10.1590/1806-9126-RBEF-2022-0101

Didactic Resources

Learning Deep Learning

Henrique F. de Arruda[*1,2], Alexandre Benatti[1], César Henrique Comin[3], Luciano da F. Costa[1]

[1]Universidade de São Paulo, Instituto de Física de São Carlos, São Carlos, SP, Brasil.
[2]ISI Foundation, Via Chisola 5, 10126, Turin, Italy.
[3]Universidade Federal de São Carlos, Departamento de Ciência da Computação, São Carlos, SP, Brasil.

Received on April 01, 2022. Revised on July 21, 2022. Accepted on July 22, 2022.

As a consequence of its capability of creating high level abstractions from data, deep learning has been effectively employed in a wide range of applications, including physics. Though deep learning can be, at first and simplistically understood in terms of very large neural networks, it also encompasses new concepts and methods. In order to understand and apply deep learning, it is important to become familiarized with the respective basic concepts. In this text, after briefly revising some works relating to physics and deep learning, we introduce and discuss some of the main principles of deep learning as well as some of its principal models. More specifically, we describe the main elements, their use, as well as several of the possible network architectures. A companion tutorial in Python has been prepared in order to complement our approach.
Keywords: Deep learning, Tutorial, Classification.

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 restricted 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 [2].

Interestingly, the ability of *abstraction* is required from humans, as a consequence of the need to prevent a level of detail that would otherwise surpasses our memory and/or processing capacity [3]. 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 *complex* (e.g. [4]), and we need to stop this subcategorization 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. [5]).

Artificial approaches to pattern recognition typically involve two main steps: (a) feature extraction; and (b) classification based on these features (e.g. [1, 6]). Figure 1 illustrates these two stages. While the former was initially human-assisted, efforts were focused on the *classification* itself. From the very beginning, neural networks were understood to provide a motivation and reference, as a consequence of the impressive power of biological nervous systems (e.g. [7]).

Interestingly, the subject of artificial neural networks (e.g. [8]) received successive waves of attention from the scientific-technologic community (for instance, respective to the Perceptron (e.g. [9]), and Hopfield networks (e.g. [10, 11]). These waves, frequently observed in scientific development, are characterized by a surge of development induced by one or more important advances, until a kind of saturation is reached as a consequence of the necessity of new major conceptual and/or technological progresses. After saturation arises,

* Correspondence email address: h.f.arruda@gmail.com

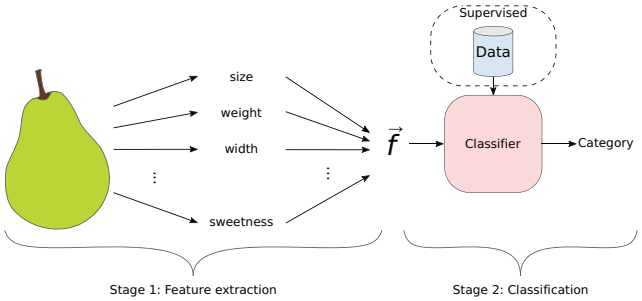

Figure 1: Scheme of classification that starts from the feature extraction step, in which information from the object (e.g. a pear) is measured, to the classification where the respective category is assigned. In this example, we show a supervised classifier. So, a database (set of training samples)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

new subsequent waves of development are induced, integrating and complementing the previous developments through new approaches and perspectives.

The great current interest in deep learning stems from the impressive performance that has been often 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 neural algorithms [12]. 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 [13]. Added to these, we also have conceptual and methodological advances such as the possibility to automatic define and extract features, as well as the incorporation of novel activation functions [13].

Aiming at creating a guideline for learning deep learning, the present text provides a brief and hopefully accessible conceptual introduction to some of the main respective aspects and developments. We present and discuss the main concepts and illustrate the respective applications. 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 paper is organized as follows. Section 2 describes the use of physics to develop deep learning as well as its applications to solve physics-related problems. Section 3 describes traditional neural networks. In Section 4, we cover the main characteristics of deep learning networks, which include novel activation functions, computer architectures, as well as some of the main deep learning architectures.

2. Deep Learning and Physics

The development of machine learning and artificial intelligence has relied on concepts from diverse areas, including but not limited to mathematics, computer science, biology, and physics. In the case of deep learning, important concepts of physics [14, 15] have often been employed. Here, we brief and non-exhaustively review some of the concepts of physics that have found their way to deep learning, and vice versa [16].

Among the various artificial neural networks developments, we have the concept of *Boltzmann Machine* (BM) [17], which is also known as stochastic Hopfield due to shared characteristics. The BM is based on the concepts of statistical physics, and its name refers to the original study Boltzmann in the areas of thermodynamics and statistics [8]. For more details, see Section 4.5.4. Variations of this type of neural network have been important to developments and applications on deep learning (e.g., [18, 19]), mainly regarding computer vision [18, 20, 21]. For instance, the *Deep Boltzmann Machine* has been applied to handwritten digit recognition (e.g. [18]).

Another interesting approach is the *Deep Lagrangian Networks* (DeLaN) [22], which considers Lagrangian mechanics in order to learn the dynamics of a system. As an example of application, [22] used *Deep Lagrangian Networks* for robot tracking control. Another work considering Lagrangian dynamics was developed aimed at applications to fiber-optic communication [23]. More specifically, a deep learning technique was used to solve a problem related to communication, in which the signal propagation is described by the nonlinear Schrödinger equation.

In addition to deep learning techniques based on physics, deep learning has also been applied to the analysis of physics-related phenomena [24, 25]. For instance, the authors of [26] proposed a deep learning model capable of learning nonlinear partial differential equations of physical dynamics. General machine learning techniques, with a focus on deep learning, have been often employed to the analysis of high-energy physics experiments. An interesting review [27] has indicated the use of deep learning in the Large Hadron Collider (LHC) collision experiments, such as the use of neural networks to assist simulations of related dynamics [28, 29]. Other examples of application of deep learning in physics include: analysis of satellite data to forecast oceanic phenomena [30], understanding the physics of extensive air showers [31], among other possibilities [24, 32].

3. Traditional Neural Networks

The idea of creating a classifier based on neurons starts from a single unity (see Fig. 2a) whose dendrites can receive the input information that is weighted by the weight matrix $\boldsymbol{W} = [w_{i,k}]$, the cell 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 provided in Section 4.3. In order

 DOI: https://doi.org/10.1590/1806-9126-RBEF-2022-0101

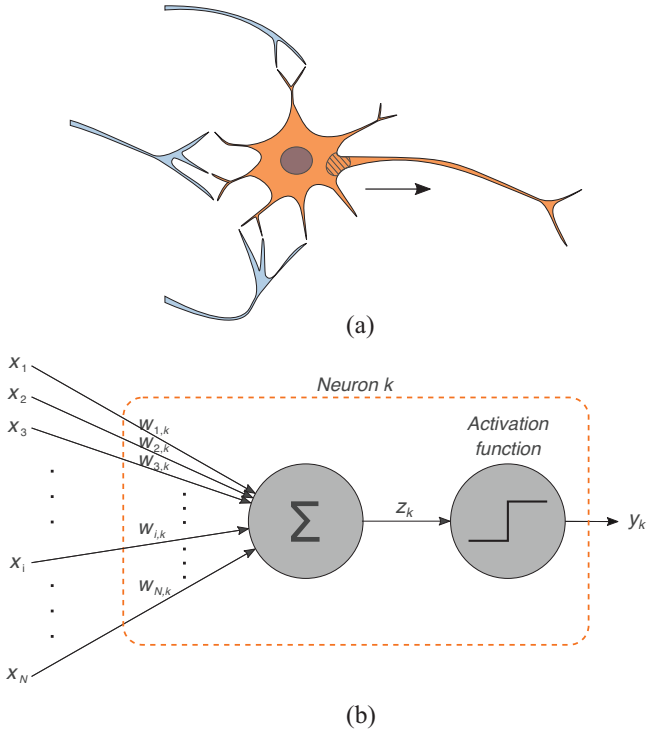

Figure 2: (a) A highly simplified biological neuron. The main parts of a neuron include: dendrites and synapses; the cellular body; the implantation cone (represented as the dashed region), in which the integration occurs; and the axons that transmit the signal to the next neuron(s). (b) A possible model of a neuron k, $k = 1, 2, \ldots, K$: The input data x_i, $i = 1, 2, \ldots, N$, come from the input layer at the left-hand side, and each of these values is multiplied by the respective weight $w_{i,k}$. These values are then summed up, yielding z_k, which is fed into the activation function, producing the output y_k.

to assign 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 updated as follows

$$w_{i,k}(n) = w_{i,k}(n-1) + \alpha\, x_i\, \epsilon, \tag{1}$$

where $w_{i,k} \in \boldsymbol{W}$, $w_{i,k}(n-1)$ are the current weights, $w_{i,k}(n)$ are the updated weights, α is the learning rate, x_i is the input data, and ϵ is the error. Interestingly, this simple methodology can classify data from two classes that are linearly separated. Figure 2 presents a highly simplified biological neuron (a) as well as a possible respective model (b).

In order to represent more general regions, sets of neurons have been considered, which are organized as a network [8]. The more straightforward manner is the use of the *Multilayer Perceptron* (MLP) [8]. In this case, the neurons are organized into three types of layers:

- Input layer: the first layer of the network (data input);
- Hidden layer: receives information from a previous and, after a sum and activation operation, transmits the data to the next 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 [8] being similar to the approach employed to a single neuron. More specifically, the training data is fed into 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 [33], which is called *universal aproximation theorem*. So, theoretically, this number of layers is enough to any problem. However, usually at least two hidden layers are used because it was found to decrease the learning time and improve the accuracy [34].

4. The Deep Learning Framework

One of the main 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 neural network and a convolutional deep learning network.

4.1. Optimization

Optimization is one of the key points of deep learning. This step consists of minimizing the loss function during neural network training. The loss function, which measures the quality of the neural 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 the location of the global minimum of this function, several methods have been considered including *gradient descent*, *stochastic gradient descent*, and *Adam* [35]. 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,

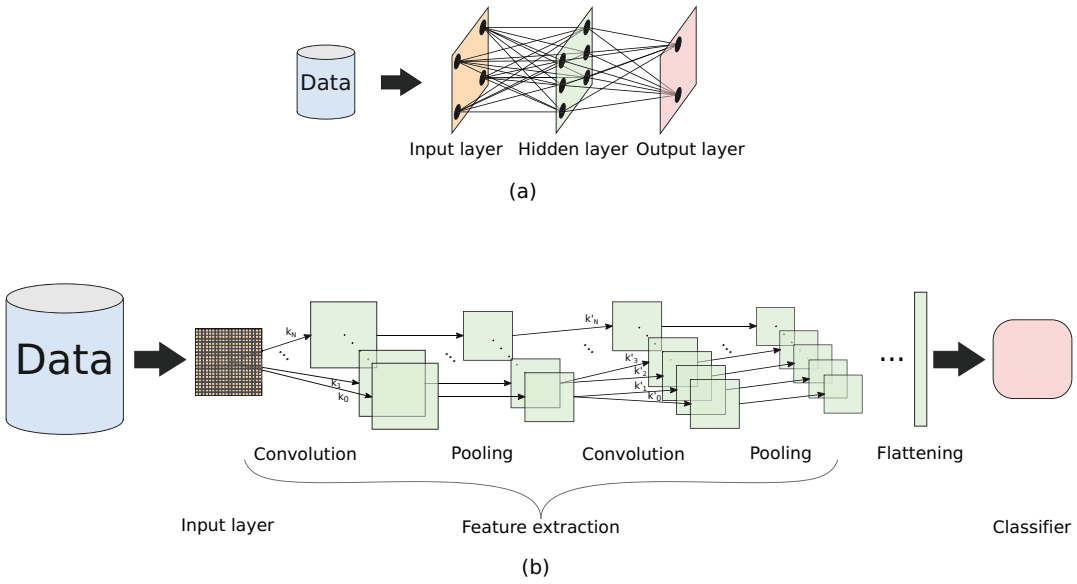

Figure 3: A simple multilayer perceptron (a), and a convolutional deep learning network (b). A convolutional network involves layers dedicated to *convolution*, *pooling*, and *flattening*. Each matrix of the convolution layer is associated with a given kernel k_i. Often a feedforward network is adopted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

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 [35]. Another advantage of Adam is that it requires relatively few memory.

4.2. 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 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 a GPGPU, which are characterized by data parallelism tasks. For instance, 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. GPGPUs typically cannot replace CPUs, but many novel approaches have become achievable with this technology, including deep learning.

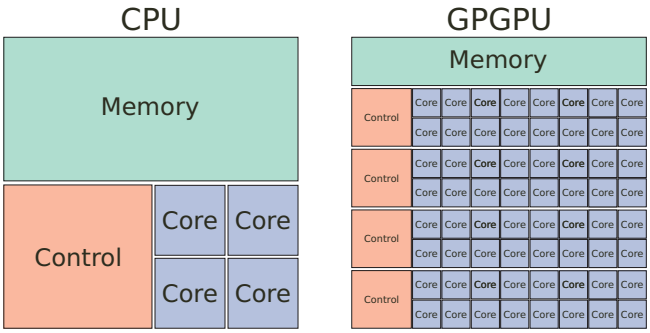

Figure 4: A simplified comparison between CPU and GPGPU. The CPU cores are more powerful and can be employed to execute complex tasks in parallel, while the GPGPU have lots of cores that are more specific to performing massive processing of data.

4.3. Activation functions

In biological neurons, the cell 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 [36], if the integrated stimulus intensity is lower than zero, the neuron is considered *unactivated*, yielding zero as result. Otherwise, the neuron returns one and is considered *activated* (see Fig. 5a). The step function is typically employed for classification of linearly separable data (e.g. [8]).

Other activation functions can also be employed. For example, we have the *sigmoid* function [13] (shown in Figure 5(b)), which can be interpreted as a probability, as it returns values between zero and one. This function

 DOI: https://doi.org/10.1590/1806-9126-RBEF-2022-0101

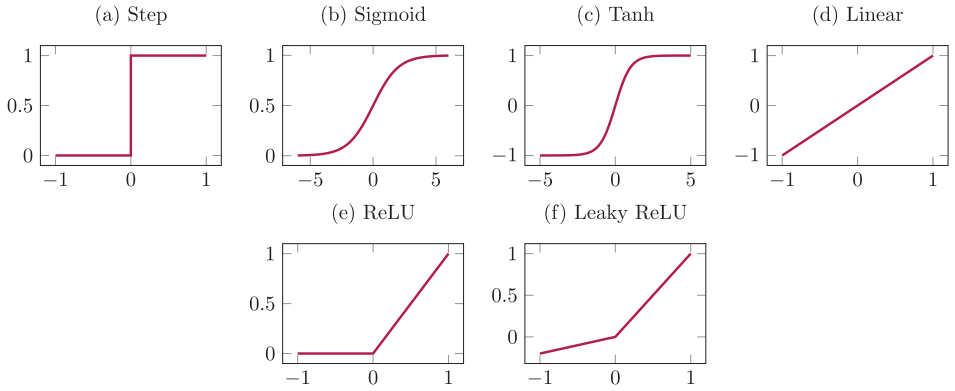

Figure 5: Examples of activation functions, used in specific neural networks-based solutions.

is defined as

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

where z is the value of the cell body (at the implantation cone) sum. Another alternative is the *hyperbolic tangent* [37], which is defined as

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

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* [13], also called *linear function*. 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 convolutional neural networks (see Section 4.5), the most common activation function is the *Rectified Linear Unit* (ReLU) [13], which is a function defined as

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

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 understood as being ease to optimize and to preserve properties having good generalization potential.

Alternatively, the Leaky Rectified Linear Units (Leaky ReLU) [38, 39] 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

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

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 [13] can be used in the last layer to deal with classification problems in which there are many distinct classes. This function is defined for each neuron k, $k = 1, 2, \ldots, K$ (see Fig. 2) as follows

$$f(z_k) = \frac{e^{z_k}}{\sum_{i=1}^{K} e^{z_i}}, \tag{6}$$

where z_i is the ith input to the respective activation function and K is the number of inputs to that function. Because the sum of the exponential values normalizes this function, it can be understood as a probability distribution.

4.4. Deep learning main aspects

This subsection briefly describes the characteristic aspects of deep learning.

4.4.1. Bias

The concept of bias consists of incorporating a fixed value, b, as input to the neural layer [40]. This value allows the activation function to adapt in order to better fit the data. Biasing can be mathematically represented as

$$y_k = f(\boldsymbol{X}^T \cdot \boldsymbol{W}_k + b), \tag{7}$$

where $\boldsymbol{X} = [x_i]$ is the input column vector, $\boldsymbol{W}_k = [w_{i,k}]$ is the column vector k derived from the weight matrix $\boldsymbol{W}$, $f(\cdot)$ is a given activation function, and y_k is the output of the neuron k.

4.4.2. 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 [41]. 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.

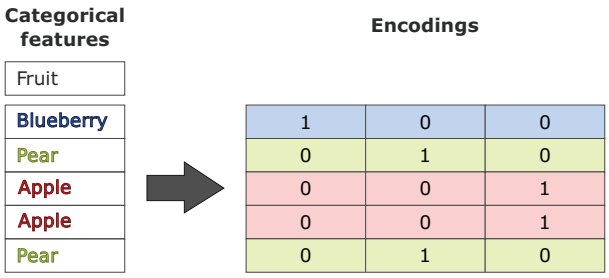

Figure 6: Example of One Hot Encoding, where the fruits are converted into a sparse matrix.

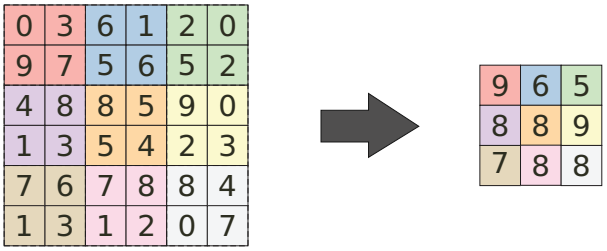

Figure 7: Example of max pooling, in which the highest number of each window is selected and assigned to the new, reduced matrix.

4.4.3. Pooling

This process is used in convolutional neural networks (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 [42]. 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.

4.4.4. Flattening

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

$$\begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,M} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,M} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N,1} & x_{N,2} & \cdots & x_{N,M} \end{bmatrix} \xrightarrow{Flattening} \begin{bmatrix} x_{1,1} \\ x_{1,2} \\ \vdots \\ x_{1,M} \\ x_{2,1} \\ x_{2,2} \\ \vdots \\ x_{2,M} \\ \vdots \\ x_{N,M} \end{bmatrix}, \quad (8)$$

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.

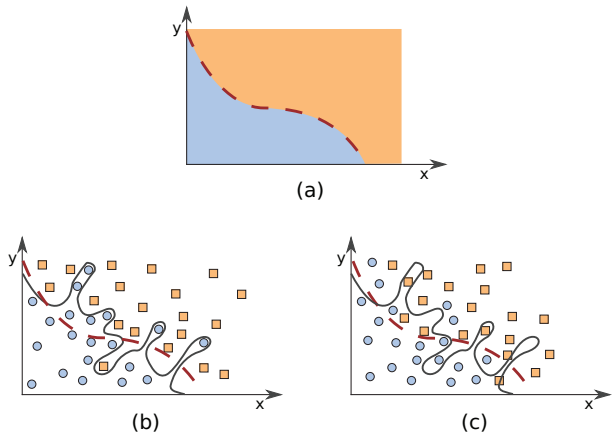

Figure 8: Example of overfitting while classifying samples from two classes, represented by blue circles and yellow squares, in the presence of noisee. The dashed lines indicate the proper separation between regions, and the black lines indicate the separation found by a classifier. This classification problem can be described by the regions as in (a), but different samplings from this problem can lead to rather different classification curves, as illustrated in (b) and (c), since the curve adheres too much to each of the noisy sampled data sets.

4.4.5. Overfitting

Overfitting (e.g. [43]) happens when the model fits, in presence of noise or original category error, 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.

4.4.6. Dropout

Dropout was proposed in order to minimize the problem of overfitting [44]. More specifically, the objective of this approach is to avoid excessive detail by replacing a percentage of the values of a given layer with zeros. The percentage of zeros is the parameter of this technique. The success of Dropout derives from the fact that the neurons do not learn too much detail of the instances of the training set. Alternatively, it can be considered that Dropout generates many different versions of a neural network, and each version has to fit the data with good accuracy.

4.4.7. Batch normalization

Batch normalization [45] is based on the idea of normalizing the input of each layer to zero mean and unit standard deviation for a given batch of data. 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

 DOI: https://doi.org/10.1590/1806-9126-RBEF-2022-0101

some cases, batch normalization can mitigate overfitting. In these cases, the use of Dropout can be unnecessary.

4.4.8. Weight regularization

Typically, overfitting leads to large values for the weights $\boldsymbol{W}$. Thus, this effect can 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 [13], such as L_1, L_2 (also called weight decay), and L_1L_2. L_1 and L_2 are defined as the sum of the absolute weights and the sum of the squared weights, respectively, and L_1L_2 employs the sum of both regularizations, which are defined as

$$L_1 = l_1 \sum_i |w_{i,k}|, \tag{9}$$

$$L_2 = l_2 \sum_i w_{i,k}^2, \tag{10}$$

$$L_1L_2 = L_1 + L_2, \tag{11}$$

where l_1 and l_2 set the amount of regularization and $w_{i,k}$ are elements of the matrix $\boldsymbol{W}$.

4.5. Types of deep learning networks

In order to deal with a variety of problems, many neural network topologies have been proposed in the literature [13]. Here, we describe some of the most used deep learning topologies, and comment on their respective applications.

4.5.1. Feedforward

Feedforward is one of the first artificial neural network topologies proposed in the literature [46]. 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 a feedforward network for binary classification, having a single neuron in the output layer.

In Deep Learning Tutorial – 1,[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 cultivars, by using a wine dataset containing chemical information about different wines. The second consists of an analysis of the same dataset, which can be classified into three classes, according to their cultivars. In this case, we show an example of the use of softmax as

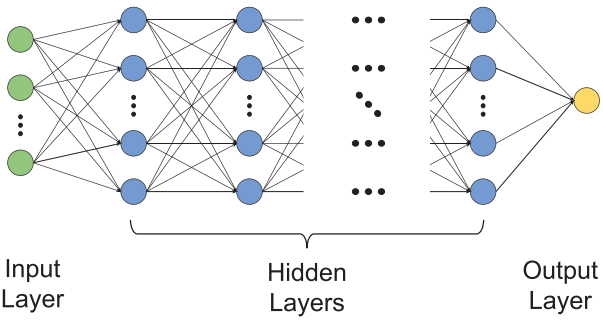

Figure 9: Example of a feedforward network that can be employed in binary classifications.

the activation function. In addition to the illustrated classification task, the feedforward networks are also employed to regression tasks.

4.5.2. Convolutional neural network

A convolutional neural network (CNN) [13] 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 need to be applied in order to counteract the large redundancy commonly found in visual data.

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

4.5.3. Recurrent neural network

Recurrent neural networks (RNN) [47] consists of a class of neural 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

[1] https://github.com/hfarruda/deeplearningtutorial/blob/master/deepLearning_feedforward.ipynb

[2] https://github.com/hfarruda/deeplearningtutorial/blob/master/deepLearning_CNN.ipynb

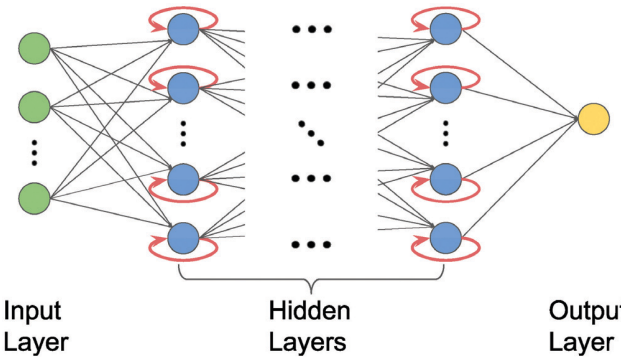

Figure 10: Example of RNN, which is similar to the structure of the feedfoward network, but with recurrent neurons.

employed to solve various problems including speech recognition, text translation, language modeling, and image captioning, among others.

One problem that can affect RNNs is called *vanishing gradient*, which consists in the gradient of loss functions vanishing to zero during the training stage [48]. In order to address this problem, it is possible to use the Long Short-Term Memory (LSTM)[3] [49]. 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,[4] we present a deep LSTM learning model able to predict Bitcoin prices along time by using the input as a temporal series.

4.5.4. Boltzmann machine

The Boltzmann machine is a stochastic neural network where all neurons are non-directly interconnected [50] (see Fig. 11a). 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* [51] is used in the training step instead of gradient descent. One disadvantage of this network in its basic configuration is its relatively high computational cost due to the high number of connections increasing exponentially with the number of neurons.

One possible solution to the problem of high computational cost is the restricted Boltzmann machine (RBM) [52], which is a variant of the Boltzmann machine (see Fig. 11b). 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

[3] More information about the LSTM can be found in http://colah.github.io/posts/2015-08-Understanding-LSTMs/.
[4] https://github.com/hfarruda/deeplearningtutorial/blob/master/deepLearning_LSTM.ipynb

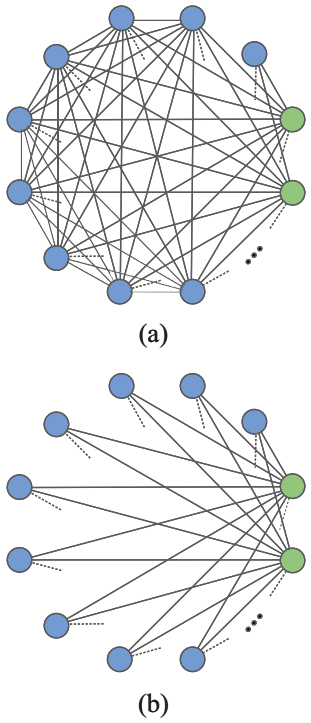

Figure 11: (a) Example of Boltzmann Machine network and (b) example of Restricted Boltzmann Machine network, in which the green and blue nodes represent visible and hidden neurons, respectively.

difference is the employed training step of RBM, which can be the contrastive divergence (CD) algorithm [52]. 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,[5] we provide an example of RBM for a recommendation system of CDs and Vinyls.

4.5.5. Autoencoders

Autoencoders consist of a deep learning model that generates a coding representation from a given data [53]. 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 training, the decoder and the output layer are removed, and the values produced by the encoder become 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.

[5] https://github.com/hfarruda/deeplearningtutorial/blob/master/deepLearning_RBM.ipynb

 DOI: https://doi.org/10.1590/1806-9126-RBEF-2022-0101

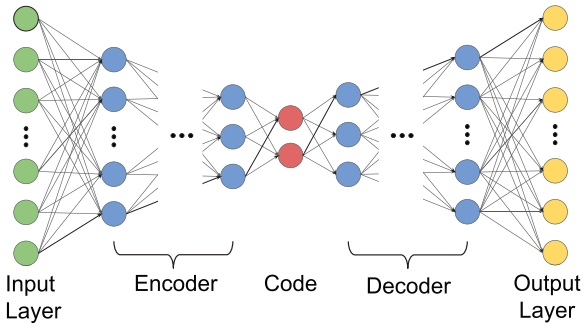

Figure 12: Example of an autoencoder network. After training, the code layer becomes the output of the network.

In order to illustrate an autoencoder application, we use the *Fashion MNIST* dataset, which comprises different types of clothes. By using the resulting codes, we project the data using a Uniform Manifold Approximation and Projection (UMAP) [54]. The code is available in Deep Learning Tutorial – 5.[6]

4.5.6. Generative adversarial networks

Generative Adversarial Networks (GANs) [55] 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 changes in network parameters [56, 57].

In Deep Learning Tutorial – 6,[7] 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.

5. 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 [40]. This approach

[6] https://github.com/hfarruda/deeplearningtutorial/blob/master/deepLearning_autoencoder.ipynb
[7] https://github.com/hfarruda/deeplearningtutorial/blob/master/deepLearning_GAN.ipynb

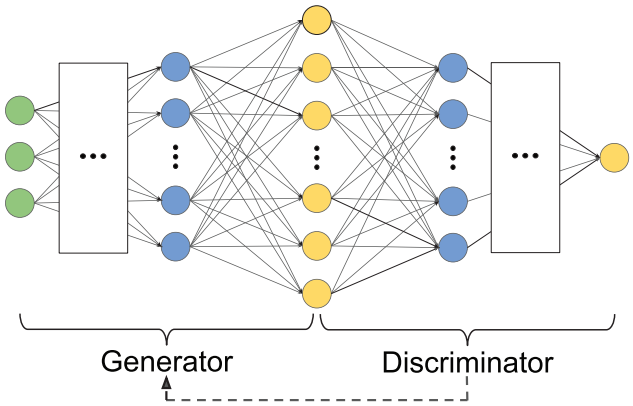

Figure 13: Example of Adversarial networks, which are divided into two parts, the generator and discriminator.

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 the training data. This process is repeated k times for all possible test groups, and then an 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 a high standard deviation suggests that the classifier has overfitted the data (ex [58]).

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.

6. Concluding Remarks

Deep learning has been used to effectively solve many problems involving classification and clustering. As such, these networks have been incorporated into the most diverse applications, ranging from automated movies subtitles to self-driving cars. In principle, deep learning structures consist of large neural 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, after briefly reviewing some interdisciplinary works involving physics and deep learning, we briefly introduced the main elements underlying deep learning, including its motivation, basic concepts, and some of the main models. These elements are

Table 1: Comparison among the models considered in this work. *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.

Models	Learning	Main Applications	Information Flow	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

particularly important for understanding a wide range of deep learning-related methods. Table 1 summarizes the revised models and some of their respective characteristics. A tutorial in Python has been 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 related literature.

Acknowledgments

Henrique F. de Arruda acknowledges FAPESP for sponsorship (grant no. 2018/10489-0, from 1st February 2019 until 31st May 2021). H. F. de Arruda also thanks Soremartec S.A. and Soremartec Italia, Ferrero Group, for partial financial support (from 1st July 2021). His funders had no role in study design, data collection, and analysis, decision to publish, or manuscript preparation. Alexandre Benatti thanks Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001. Luciano da F. Costa thanks CNPq (grant no. 307085/2018-0) and FAPESP (proc. 15/22308-2) for sponsorship. César H. Comin thanks FAPESP (Grant Nos. 2018/09125-4 and 2021/12354-8) for financial support. This work has been supported also by FAPESP grants 11/50761-2 and 15/22308-2.

References

[1] L.F. Costa, *Modeling: The human approach to science (cdt-8)*, available in: https://www.researchgate.net/publication/333389500_Modeling_The_Human_Approach_to_Science_CDT-8, accessed in 06/06/2019.

[2] E.B. Goldstein and J. Brockmole, *Sensation and perception* (Cengage Learning, Belmont, 2016).

[3] R.G. Cook and J.D. Smith, Psychological Science **17**, 1059 (2006).

[4] L.F. Costa, *Quantifying complexity (cdt-6)*, available in: https://www.researchgate.net/publication/332877069_Quantifying_Complexity_CDT-6, accessed in 06/06/2019.

[5] L.F. Costa and R.M. Cesar Jr, *Shape analysis and classification: theory and practice* (CRC Press, Inc., Boca Raton, 2000).

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

[7] F.R. Monte Ferreira, M.I. Nogueira and J. DeFelipe, Frontiers in neuroanatomy **8**, 1 (2014).

[8] S. Haykin, in: *Neural networks and learning machines* (Pearson Education, India, 2009), 3 ed., v. 10.

[9] I. Stephen, IEEE Transactions on neural networks **50**, 179 (1990).

[10] J.D. Keeler, Cognitive Science **12**, 299 (1988).

[11] B. Xu, X. Liu and X. Liao, Computers & Mathematics with Applications **45**, 1729 (2003).

[12] S. Dutta, Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery **8**, e1257 (2018).

[13] I. Goodfellow, Y. Bengio and A. Courville, *Deep learning* (MIT press, Cambridge, 2016).

[14] N. Thuerey, P. Holl, M. Mueller, P. Schnell, F. Trost and K. Um, arXiv:2109.05237 (2021).

[15] A. Tanaka, A. Tomiya and K. Hashimoto, *Deep Learning and Physics* (Springer, Singapore, 2021).

[16] L. Zdeborva, Nature Physics **16**, 602 (2020).

[17] D.E. Rumelhart, G.E. Hinton, J.L. McClelland, in: *Parallel distributed processing: Explorations in the microstructure of cognition*, edited by D.E. Rumelhart and J.L. McClelland (MIT Press, Cambridge, 1986).

[18] R. Salakhutdinov and H. Larochelle, in: *Proceedings of the thirteenth international conference on artificial intelligence and statistics* (Sardinia, 2010).

[19] M.H. Amin, E. Andriyash, J. Rolfe, B. Kulchytskyy, and R. Melko, Physical Review X **8**, 021050 (2018).

[20] N. Srivastava and R.R. Salakhutdinov, Advances in neural information processing systems **25** (2012).

[21] I. Goodfellow, M. Mirza, A. Courville and Y. Bengio, in: *Proceedings of Advances in Neural Information Processing Systems 26* (Lake Tahoe, 2013).

[22] M. Lutter, C. Ritter and J. Peters, arXiv:1907.04490 (2019).

[23] C. Häger and H.D. Pfister, IEEE Journal on Selected Areas in Communications **39**, 280 (2020).

[24] P. Sadowski and P. Baldi, in: *Braverman Readings in Machine Learning. Key Ideas from Inception to Current*

State, edited by L. Rozonoer, B. Mirkin and I. Muchnik (Springer, Boston, 2018).

[25] M. Erdmann, J. Glombitza, G. Kasieczka and U. Klemradt, *Deep Learning for Physics Research* (World Scientific, Singapore, 2021).

[26] M. Raissi, The Journal of Machine Learning Research **19**, 932 (2018).

[27] D. Guest, K. Cranmer and D. Whiteson, Annual Review of Nuclear and Particle Science **68**, 161 (2018).

[28] T.A. Le, A.G. Baydin and F. Wood, in: *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics PMLR 54* (Fort Lauderdale, 2017).

[29] A.G. Baydin, L. Shao, W. Bhimji, L. Heinrich, L. Meadows, J. Liu, A. Munk, S. Naderiparizi, B. Gram-Hansen, G. Louppe, *et al.*, in: *Proceedings of the international conference for high performance computing, networking, storage and analysis* (Denver, 2019).

[30] G. Zheng, X. Li, R.H. Zhang and B. Liu, Science advances **6**, eaba1482 (2020).

[31] A. Guillen, A. Bueno, J. Carceller, J. Martınez-Velazquez, G. Rubio, C.T. Peixoto and P. Sanchez-Lucas, Astroparticle Physics **111**, 12 (2019).

[32] M. Reichstein, G. Camps-Valls, B. Stevens, M. Jung, J. Denzler, N. Carvalhais and Prabhat, Nature **566**, 195 (2019).

[33] R. Hecht-Nielsen, in: *Proceedings of the international conference on Neural Networks* (New York, 1987).

[34] M.M. Poulton, in: *Handbook of Geophysical Exploration: Seismic Exploration* (Elsevier, Amsterdam, 2001), v. 30.

[35] D.P. Kingma and J. Ba, arXiv:1412.6980 (2014).

[36] A.K. Jain, J. Mao and K. Mohiuddin, Computer **29**, 31 (1996).

[37] P. Sibi, S.A. Jones and P. Siddarth, Journal of Theoretical and Applied Information Technology **47**, 1264 (2013).

[38] A.L. Maas, A.Y. Hannun and A.Y. Ng, in: *Proceeding International Conference on Machine Learning* (Atlanta, 2013).

[39] B. Xu, N. Wang, T. Chen and M. Li, arXiv:1505.00853 (2015).

[40] C.M. Bishop and N.M. Nasrabadi, *Pattern recognition and machine learning* (Springer, Berlim, 2006), v. 4.

[41] A. Deshpande and M. Kumar, *Artificial intelligence for big data: Complete guide to automating big data solutions using artificial intelligence techniques* (Packt Publishing Ltd, Birmingham, 2018).

[42] M. Cheung, J. Shi, O. Wright, L.Y. Jiang, X. Liu and J.M. Moura, IEEE Signal Processing Magazine **37**, 139 (2020).

[43] X. Ying, in: *2018 International Conference on Computer Information Science and Application Technology – v. 1168* (Daqing, 2019).

[44] G.E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever and R.R. Salakhutdinov, arXiv:1207.0580 (2012).

[45] S. Ioffe and C. Szegedy, in: *Proceedings of the 32nd International Conference on Machine Learning* (Mountain View, 2015).

[46] J. Schmidhuber, Neural networks **61**, 85 (2015).

[47] D.E. Rumelhart, G.E. Hinton and R.J. Williams, Nature **323**, 533 (1986).

[48] Y. Bengio, P. Simard and P. Frasconi, IEEE transactions on neural networks **5**, 157 (1994).

[49] S. Hochreiter and J. Schmidhuber, Neural computation **9**, 1735 (1997).

[50] G.E. Hinton, S. Osindero and Y.W. Teh, Neural computation **18**, 1527 (2006).

[51] E. Aarts, J. Korst, *Simulated annealing and Boltzmann machines: a stochastic approach to combinatorial optimization and neural computing* (John Wiley & Sons, Inc., Hoboken, 1989).

[52] G.E. Hinton, Neural computation **14**, 1771 (2002).

[53] P. Baldi, in: *Proceedings of ICML workshop on unsupervised and transfer learning, PMLR 27* (Bellevue, 2012).

[54] L. McInnes, J. Healy and J. Melville, arXiv:1802.03426 (2018).

[55] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville and Y. Bengio, in: *Advances in neural information processing systems 27*, edited by Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence and K.Q. Weinberger (NeurIPS Proceedings, Montreal, 2014).

[56] K. Roth, A. Lucchi, S. Nowozin and T. Hofmann, in: *Advances in Neural Information Processing Systems 30*, edited by I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan and R. Garnett (NeurIPS Proceedings, Montreal, 2014).

[57] L. Metz, B. Poole, D. Pfau and J. Sohl-Dickstein, in: *Proceedings of International Conference on Learning Representations* (San Juan, 2016).

[58] G.C. Cawley and N.L. Talbot, Journal of Machine Learning Research **11**, 2079 (2010).