# Neural Networks Foundations

Artificial neural networks (briefly, nets) represent a class of machine learning models, loosely inspired by studies about the central nervous systems of mammals. Each net is made up of several interconnected neurons, organized in layers, which exchange messages (they fire, in jargon) when certain conditions happen. Initial studies were started in the late 1950s with the introduction of the perceptron (for more information, refer to the article: The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain, by F. Rosenblatt, Psychological Review, vol. 65, pp. 386 - 408, 1958), a two-layer network used for simple operations, and further expanded in the late 1960s with the introduction of the backpropagation algorithm, used for efficient multilayer networks training (according to the articles: Backpropagation through Time: What It Does and How to Do It, by P. J. Werbos, Proceedings of the IEEE, vol. 78, pp. 1550 - 1560, 1990, and A Fast Learning Algorithm for Deep Belief Nets, by G. E. Hinton, S. Osindero, and Y. W. Teh, Neural Computing, vol. 18, pp. 1527 - 1554, 2006). Some studies argue that these techniques have roots dating further back than normally cited (for more information, refer to the article: Deep Learning in Neural Networks: An Overview, by J. Schmidhuber, vol. 61, pp. 85 - 117, 2015). Neural networks were a topic of intensive academic studies until the 1980s, when other simpler approaches became more relevant. However, there has been a resurrection of interest starting from the mid-2000s, thanks to both a breakthrough fast-learning algorithm proposed by G. Hinton (for more information, refer to the articles: The Roots of Backpropagation: From Ordered Derivatives to Neural Networks and Political Forecasting, Neural Networks, by S. Leven, vol. 9, 1996 and Learning Representations by Backpropagating Errors, by D. E. Rumelhart, G. E. Hinton, and R. J. Williams, vol. 323, 1986) and the introduction of GPUs, roughly in 2011, for massive numeric computation.

These improvements opened the route for modern deep learning, a class of neural networks characterized by a significant number of layers of neurons, which are able to learn rather sophisticated models based on progressive levels of abstraction. People called it deep with 3-5 layers a few years ago, and now it has gone up to 100-200.

This learning via progressive abstraction resembles vision models that have evolved over millions of years in the human brain. The human visual system is indeed organized into different layers. Our eyes are connected to an area of the brain called the **visual cortex V1**, which is located in the lower posterior part of our brain. This area is common to many mammals and has the role of discriminating basic properties and small changes in visual orientation, spatial frequencies, and colors. It has been estimated that V1 consists of about 140 million neurons, with 10 billion connections between them. V1 is then connected with other areas V2, V3, V4, V5, and V6, doing progressively more complex image processing and recognition of more sophisticated concepts, such as shapes, faces, animals, and many more. This organization in layers is the result of a huge number of attempts tuned over several 100 million years. It has been estimated that there are ~16 billion human cortical neurons, and about 10%-25% of the human cortex is devoted to vision (for more information, refer to the article: The Human Brain in Numbers: A Linearly Scaled-up Primate Brain, by S. Herculano-Houzel, vol. 3, 2009). Deep learning has taken some inspiration from this layer-based organization of the human visual system: early artificial neuron layers learn basic properties of images, while deeper layers learn more sophisticated concepts.

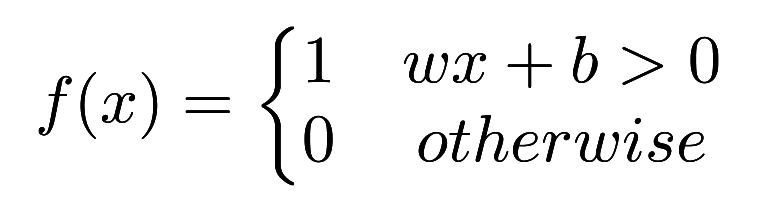
This book covers several major aspects of neural networks by providing working nets coded in Keras, a minimalist and efficient Python library for deep learning computations running on the top of either Google's TensorFlow (for more information, refer to <https://www.tensorflow.org/>) or University of Montreal's Theano (for more information, refer to <http://deeplearning.net/software/theano/>) backend. So, let's start.

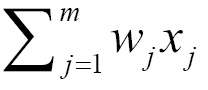
In this chapter, we will cover the following topics:

* Perceptron
* Multilayer perceptron
* Activation functions
* Gradient descent
* Stochastic gradient descent
* Backpropagation

# Perceptron

The perceptron is a simple algorithm which, given an input vector x of m values (x1, x2, ..., xn) often called input features or simply features, outputs either 1 (yes) or 0 (no). Mathematically, we define a function:



Here, w is a vector of weights, wx is the dot product , and b is a bias. If you remember elementary geometry, wx + b defines a boundary hyperplane that changes position according to the values assigned to w and b. If x lies above the straight line, then the answer is positive, otherwise it is negative. Very simple algorithm! The perception cannot express a maybe answer. It can answer yes (1) or no (0) if we understand how to define w and b, that is the training process that will be discussed in the following paragraphs.

# The first example of Keras code

The initial building block of Keras is a model, and the simplest model is called **sequential**. A sequential Keras model is a linear pipeline (a stack) of neural networks layers. This code fragment defines a single layer with 12 artificial neurons, and it expects 8 input variables (also known as features):

from keras.models import Sequential  
model = Sequential()  
model.add(Dense(12, input\_dim=8, kernel\_initializer='random\_uniform'))

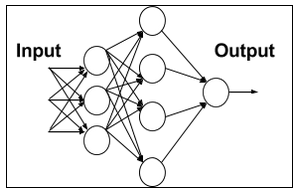
Each neuron can be initialized with specific weights. Keras provides a few choices, the most common of which are listed as follows:

* random\_uniform: Weights are initialized to uniformly random small values in (-0.05, 0.05). In other words, any value within the given interval is equally likely to be drawn.
* random\_normal: Weights are initialized according to a Gaussian, with a zero mean and small standard deviation of 0.05. For those of you who are not familiar with a Gaussian, think about a symmetric bell curve shape.
* zero: All weights are initialized to zero.

A full list is available at <https://keras.io/initializations/>.

# Multilayer perceptron — the first example of a network

In this chapter, we define the first example of a network with multiple linear layers. Historically, perceptron was the name given to a model having one single linear layer, and as a consequence, if it has multiple layers, you would call it **multilayer perceptron** (**MLP**). The following image represents a generic neural network with one input layer, one intermediate layer and one output layer.



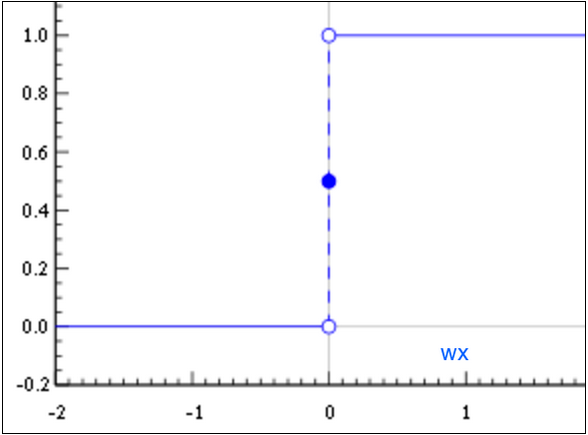
In the preceding diagram, each node in the first layer receives an input and fires according to the predefined local decision boundaries. Then the output of the first layer is passed to the second layer, the results of which are passed to the final output layer consisting of one single neuron. It is interesting to note that this layered organization vaguely resembles the patterns of human vision we discussed earlier.

The net is dense, meaning that each neuron in a layer is connected to all neurons located in the previous layer and to all the neurons in the following layer.

# Problems in training the perceptron and a solution

Let's consider a single neuron; what are the best choices for the weight w and the bias b? Ideally, we would like to provide a set of training examples and let the computer adjust the weight and the bias in such a way that the errors produced in the output are minimized. In order to make this a bit more concrete, let's suppose we have a set of images of cats and another separate set of images not containing cats. For the sake of simplicity, assume that each neuron looks at a single input pixel value. While the computer processes these images, we would like our neuron to adjust its weights and bias so that we have fewer and fewer images wrongly recognized as non-cats. This approach seems very intuitive, but it requires that a small change in weights (and/or bias) causes only a small change in outputs.

If we have a big output jump, we cannot progressively learn (rather than trying things in all possible directions—a process known as exhaustive search—without knowing if we are improving). After all, kids learn little by little. Unfortunately, the perceptron does not show this little-by-little behavior. A perceptron is either 0 or 1 and that is a big jump and it will not help it to learn, as shown in the following graph:



We need something different, smoother. We need a function that progressively changes from 0 to 1 with no discontinuity. Mathematically, this means that we need a continuous function that allows us to compute the derivative.