Photonic Artificial Neural Networks:
All-Optical Activation Function

Davide Bazzanella

Department of Physics, University of Trento

January 22, 2018

Contents

Introduction **1**

1 Artificial Neural Networks **3**

1.1 Basis of Neural Networks . . . 3

1.2 Working Principles of ANNs . . . 4

1.2.1 Learning Process . . . 5

1.2.2 Validation Process . . . 7

1.2.3 Testing Process . . . 7

1.2.4 Datasets . . . 7

1.3 Feedforward NN . . . 7

1.3.1 Other Types of NNs . . . 8

1.4 Real-Life Examples . . . 9

2 Photonics applied to ANNs **11**

2.1 Weighted sum of inputs . . . 11

2.2 Nonlinear activation function . . . 11

2.2.1 Simulations . . . 11

3 Samples, setup and measurements **13**

3.1 The samples . . . 13

3.2 The setup . . . 13

3.3 Results . . . 13

Conclusions **15**

Bibliography **17**

Aknowledgements **19**

Introduction

WHAT and WHY? BACKUP project, why silicon photonics (CMOS compatibility)

The formalization of the von Neumann architecture (1945) and the development of modern computers in the second half the 20th century allowed a great improvement in many scientific and technological areas. The technological progress led to [...]

Now, on the verge of the era of artificial intelligence (AI), many breakthroughs in complex of AIs are happening. This is mainly due to the immense amount of data fed to AI algorithms. For this reason however, the algebraic calculus which is carried out so efficiently by computers is not enough anymore.

Under the

Chapter 1

Artificial Neural Networks

Artificial Neural Networks (ANNs) are computational systems which elaborate information likewise biological neural networks (animal brains).

ANNs can be either simulated on computers or built on specific hardware designed ad hoc. Both options can be modeled by

For simplicity in this chapter I will consider only logical models of artificial neural networks, which are therefore closer to simulated ANN than hardware ANN.

This chapter is intended as an introduction to the world of artificial neural network, and focuses on the main aspects and types of neural networks, without claiming to be comprehensive.

1.1 Basis of Neural Networks

A neural network is a collection of processing elements, or nodes, interconnected in an arbitrary topology. From its input nodes, the network accepts information, which will propagate into the inner nodes through the interconnections and will get elaborated at each node. At the end of the network, there will be a number of output nodes, with the task of reading a portion of the inner nodes. The inner nodes are also called hidden, because they are not meant to be accessible to the external world. A generic scheme of such network is shown in Figure 1.1 on this page.

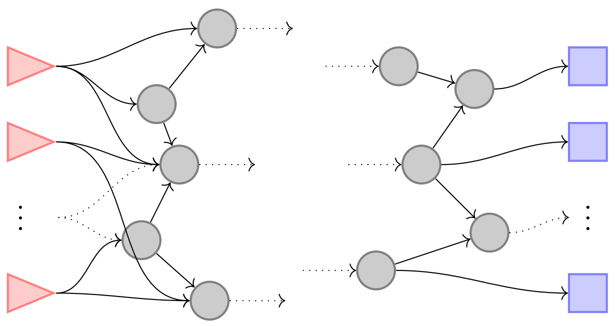

Figure 1.1: Generic scheme of a neural network. Triangles (red) are input nodes, circles (grey) are inner nodes, and squares (blue) are output nodes. Interconnections among nodes are represented by arrows: continuous when both elements are drawn, and dotted otherwise. I chose a non-standard representation to emphasize the nonlinear nodes, which will use a nonlinear activation function.

Each node can operate in the same way of the others or in a completely different manner, depending on the type of neural network. The operation of nodes resembles that of animal neurons: various input gets collected and elaborated together to obtain an output, which will become one of the many inputs for subsequent neurons/nodes. Specifically, the most used model for neurons is the McCulloch–Pitts (MCP) neuron. It is divided into two parts, as shown in Figure 1.2 on the current page: the first part is a weighted sum of the inputs, while the second part is given by the so called activation function.

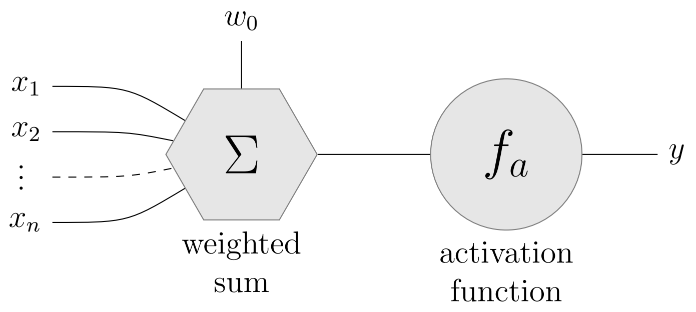

Figure 1.2: Generic node representation. x-values are inputs, y-values are outputs, w_0 is the bias.

The node is described mathematically by Equation 1.1

$$y = f_a\left(w_0 + \sum_{i=1}^{n} w_i x_i\right), \tag{1.1}$$

where f_a is the activations function, evaluated on the sum of the input x_i weighted with w_i, plus a bias w_0.

Each node accepts values at its inputs and produces an output accordingly. However, the output depends also on the node's parameters: the weights and the bias, which are usually changed outside the operation phase of the neural network, as I will explain in Section 1.2 later on this page.

Moreover it is mandatory for the activation function $f_a(\cdot)$ to be nonlinear, because otherwise a collection of nodes will result in just a weighted sum of its inputs. Two examples of nonlinear function are shown below in Figure 1.3.

One can distinguish at least three type of nodes in every neural network: input, inner/hidden, and output nodes. Input nodes take one input value, from the outside of the neural network, and pass it on to the inner nodes unchanged. Inner/hidden nodes take many inputs and generate an output through the activation function. Output nodes, similarly to input nodes, take one input value, from the inside of the NN, and pass it on to the outside.

This is not the orthodox description, however it is consistent with the idea of functional *black box*, in which input and output are the only visible nodes, while the other are hidden inside. Therefore I can easily distinguish the nodes which will use a nonlinear activation function.
WHAT ARE NEURAL NETWORK GOOD FOR?
WHEN DO I TALK ABOUT THAT?
E.G. NN ARE GOOD FOR PATTERN RECOGNITION.

1.2 Working Principles of ANNs

Because of its topology, each neural network will behave in a different manner from other neural networks with diverse, or even similar, arrangements of nodes. Moreover the same neural network will perform a certain task better or worse also depending on how inputs are weighted at each hidden node, and normally those parameters are initialized with a random

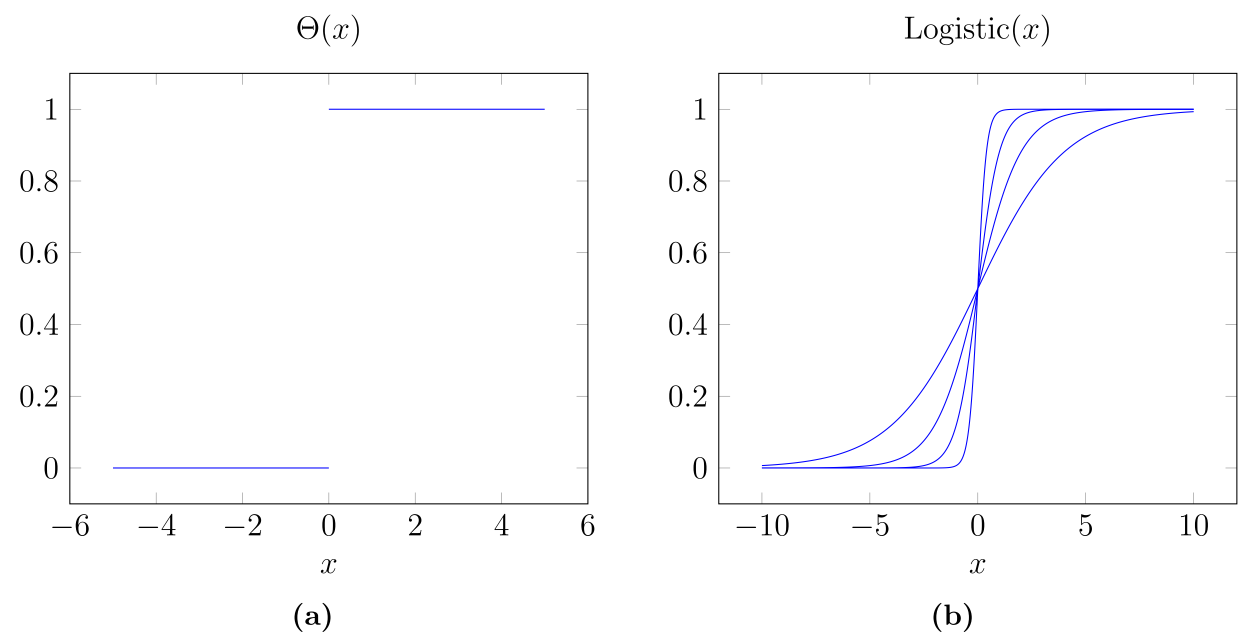

Figure 1.3: Examples of activation function: (1.3a) is the well-known step function, or Heaviside Θ, (1.3b) depicts a few functions from the family of the Logistic functions.

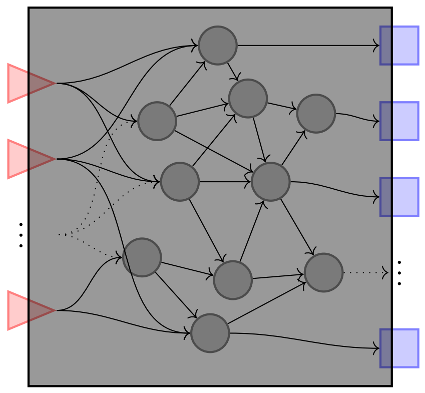

Figure 1.4: Representation of the black-box concept

value at the creation of the network. For this reason, before a neural network is considered ready to perform a task, it usually must go through three training stages: learning phase, validation phase, and testing phase. Every one of these stages are meant to prepare the network to work as required from the designer.

1.2.1 Learning Process

During the learning process the neural network is run on a set of known inputs x, each paired with its correct answer y, or target, in a second set of data. The neural network will produce at the output a third set $\hat{y}$, which should be as close as possible to the correct answers, when the network works properly. However this happens rarely, if ever, and a change in the way data is elaborated becomes necessary. The usual **??** way is to keep the same topology, but tweak the weights that connect the hidden nodes together. On account of this need, one needs to quantify the distance of the predicted result of the artificial neural network from the correct answer. This is made by means of the loss function.

Loss function

The loss function $L(y, \hat{y})$ evaluates the difference between the predicted and the correct answer. Usually, this quantity is linked to the geometrical distance between the predicted output and the target $|\hat{y} - y|$. The most common loss function is the *mean-square error*. Assuming to have an input set of N examples paired with the same number of targets, and that the outputs and the targets are composed by C values, or classes, the function becomes:

$$L(y, \hat{y}) = f_{MSE}(y, \hat{y}) = \frac{1}{N} \sum_{n=1}^{N} \sum_{i=1}^{C} \left(\hat{y}_{n,i} - y_{n,i}\right)^2 , \tag{1.2}$$

where each example in the set is subtracted to its target and then squared. Finally the mean of all squares gives the expected result.

Another commonly used function is the cross-entropy loss (also known as negative log likelihood),

$$L(y, \hat{y}) = f_{CEL}(y, \hat{y}) = -\frac{1}{N} \sum_{n=1}^{N} \sum_{i=1}^{C} y_{n,i} \log\left(\hat{y}_{n,i}\right) , \tag{1.3}$$

which expects positive values at the input. Hence the error $-y \log(\hat{y})$, quantified for each element in each example, is always a positive number. The mean over all examples in the set returns the results.

Alternatively, variations of the previous methods are given by not taking the sum of the examples in place of the mean, or by calculating a loss function for each example instead of evaluating it for the whole set.

Weights Update Process

The weights update process is a difficult task, and probably the most computationally expensive one in running a neural network. There is a variety of methods to chose from, depending on the type of artificial network and the resources available.

A widely used algorithm is the gradient descent, from its most simple version to more complex variations such as stochastic gradient descent (SGD). This method updates the weights by subtracting a value proportional to the gradient of the loss function in respect to the weights themselves times a positive factor called *learning rate*, as shown below.

$$w_i\big|_{n+1} = w_i\big|_n - lr \cdot \frac{\partial L}{\partial\, w_i\big|_n} \tag{1.4}$$

where $w_i\big|_n$ are the current weights, lr is the learning rate, $\frac{\partial L}{\partial\, w_i\big|_n}$ is the first derivative of the loss function in respect to the i-th weight at the current step, and $w_i\big|_{n+1}$ are the updated weights. This method is equivalent to minimize the error on the loss function, by following the gradient $\bigtriangledown_w L$. This vector lives in the multidimensional space of the loss function $L : \mathbb{R}^W \mapsto \mathbb{R}$, where parameters and variables are swapped.

The most efficient **??** algorithm is called *backpropagation*: it computes the first derivative of the loss function L in respect to all the parameters of the network, the weights, starting from the end of the artificial network and going backward toward the input, hence the name backpropagation. Since the number of connections between nodes might be even order of magnitude bigger than the number of nodes, it is simple to understand how large networks are computationally expensive to train.

1.2.2 Validation Process

1.2.3 Testing Process

At the end, there is the process of testing the artificial neural network. Ideally the network is tested on a new set of data, for which the target results are known, similarly to the preceding phases. This time, however, the predicted outputs are compared to the correct answers to obtain an overall value for the correctness, often expressed in percentage.

1.2.4 Datasets

Since there are three phases of preparation for any artificial neural network, there must be an appropriate number of examples to feed to it.
HOW DO I DIVIDE A DATASET?
WHAT HAPPENS WHEN THERE ARE TOO FEW EXAMPLES?
AND WHEN THERE ARE TOO MANY?

1.3 Feedforward NN

The first and most simple type of neural network is called Feedforward. In this kind of neural network, nodes are divided into groups called *layers*. A layer is a collection of nodes that accepts inputs from a preceding group and generate as many outputs as the number of nodes in the layer. Each layer of a Feedforward neural network is connected in series with the others, except of input layer at the beginning and the output layer at the end. As for the single nodes, the inner layer are called hidden, because usually not accessible.

The information travels from the input to the output and gets elaborated from each hidden layer: there are no connection between nodes of the same layer, nor loops or feedback between layers. Depending on the topology of the network, there might be more or less layers, each composed by the same or a different number of nodes. Moreover the connection between the layers might be complete, i.e. each node in the layer accepts each input of the preceding layer, in that case the layer is said to be *fully connected*, or sparse as in the case of convolutional layers (see Section 1.3).

Perceptron

The most naive topology of a Feedforward neural network is given by the so called *Perceptron*. The Perceptron dates back to the 1957, when the homonym *Perceptron algorithm* was software implemented by Frank Rosenblatt on a computer (IBM 704) and only subsequently in hardware as the *Mark 1 perceptron*[1], [2]. The graph of a generic (single layer) perceptron ANN is shown in Figure 1.5 below.

By adding more than one hidden perceptron layer to the neural network, one obtain the so called *Multi-Layer Perceptron* (MLP). This allows for more computational complexity **??**. When the total number of layer is more than two, the network is called *deep*. A deep MLP is shown in Figure 1.6 on the next page. In principle any shape is possible, i.e. each layer could have a different number of nodes, however often the layers at the beginning are wider than the layer at the end of the network **??**. Besides the shape, in literature a perceptron is almost always considered fully connected **??**.

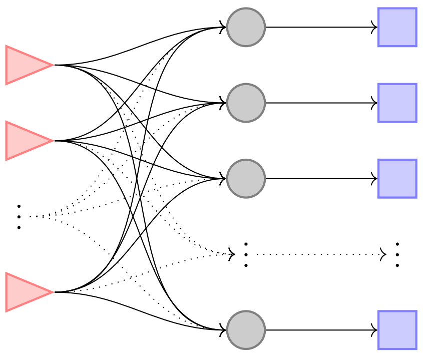

Figure 1.5: Perceptron type neural network: in this representation the perceptron has n inputs and m outputs as well as a hidden layer with m nodes. Colors, shape and styles are the same as in Figure 1.1 on page 3.

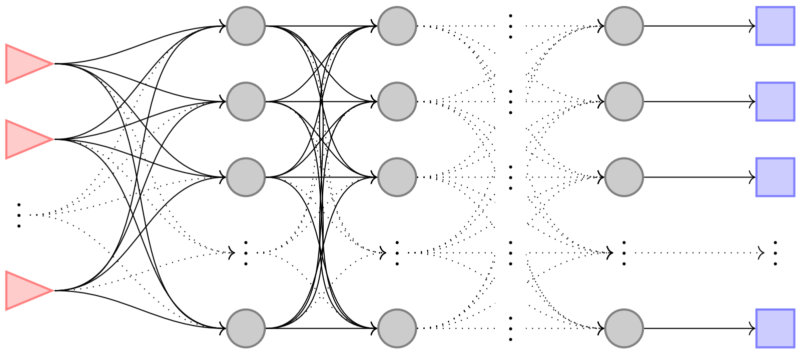

Figure 1.6: Deep Multi-Layer Perceptron (MLP), fully connected.

Other Feedforward NNs

Autoencoder neural networks are feedforward networks in which the output nodes are as many as the input nodes. The purpose of this kind of network is to reconstruct its own inputs.

Probabilistic

Time delay

Convolutional networks are inspired to the visual cortex, in which neurons are not fully connected their inputs but only to a restricted region. Convolutional neural networks are a type of feedforward network conceived to recognize images without being misled by distortions such as translation, skewing, or scaling. In this kind of network the input is often represented by a 2D matrix, instead of a 1D vector. It is usually composed by many layers, the prevalent kind is the convolutional one. This layer performs a two-dimensional convolution over the input matrix of a second 2D matrix of weights, called *feature map*. Thus, each node of the layer operates on a restricted region to understand if a feature is present or not. Commonly the operating regions are overlapping and the feature map is shared among the nodes in the same layer.

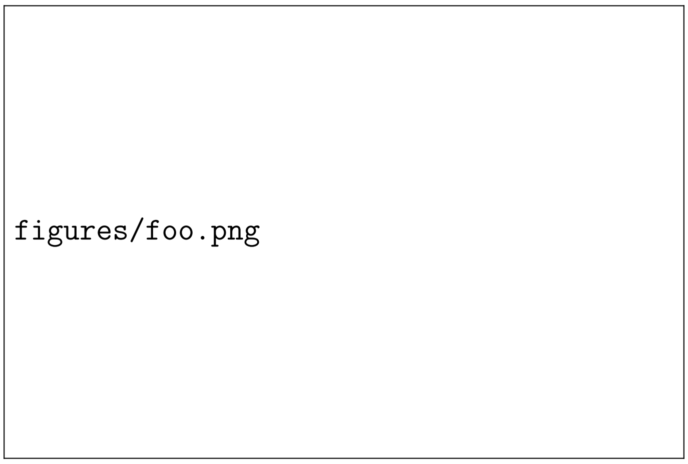

Figure 1.7: image/scheme of the conv NN

1.3.1 Other Types of NNs

Recurrent NN

Reservoir NN

Modular NN

Spiking NN

other type of neuron model: Hodgkin–Huxley (H-H) model `https://en.wikipedia.org/wiki/Binding_neuron`

1.4 Real-Life Examples

Chapter 2

Photonics applied to ANNs

How do I intend to create a hardware photonic ANN node?

2.1 Weighted sum of inputs

This has already been demonstrated and integrated widely, so it will not be the focus of this work.

2.2 Nonlinear activation function

On the other hand, a photonic nonlinear activation function has not yet been found. This is where the focus of my work will be.

2.2.1 Simulations

Chapter 3

Samples, setup and measurements

3.1 The samples

I could say that in the time this work was done there would have not been enough time to design and produce an ad hoc device. The aim of this thesis is to produce a proof of concept, to answer the question of feasibility.

The samples on which this work is based have been provided by the IRIS project. Specifically I had a few different structures available: from single rings resonators to the full matrix, each accessible via grating couplers. My choice was a system of intermediate complexity: a simple waveguide, coupled to eight drop channels by a single or a couple of ring resonators each. Moreover, these samples were provided with thermo-electric pads to heat the rings and effectively tune their resonance. The final choice was to study the *mini-matrix* in which the coupling mechanism was provided by single ring resonators, because it was considered simpler.

figures/foo.png

Figure 3.1: image/scheme of the minimatrix

3.2 The setup

3.3 Results

Conclusion

Bibliography

[1] R. Frank, “The perceptron a perceiving and recognizing automaton”, *tech. rep., Technical Report 85-460-1*, 1957.

[2] F. Rosenblatt, “The perceptron: A probabilistic model for information storage and organization in ...”, *Psychological Review*, vol. 65, no. 6, pp. 386–408, 1958, ISSN: 1939-1471(Electronic);0033-295X(Print). DOI: 10.1037/h0042519. arXiv: arXiv:1112.6209. [Online]. Available: http://psycnet.apa.org/journals/rev/65/6/386.pdf%7B%5C%%7D5Cnpapers://c53d1644-cd41-40df-912d-ee195b4a4c2b/Paper/p15420.

Aknowledgements