Alma Mater Studiorum · University of Bologna

School of Science
Department of Physics and Astronomy
Master Degree in Physics

OPTIMIZATION AND APPLICATIONS OF DEEP LEARNING ALGORITHMS FOR SUPER-RESOLUTION IN MRI

Supervisor:
Prof. Gastone Castellani

Co-supervisor:
Dr. Nico Curti

Submitted by:
Mattia Ceccarelli

Academic Year 2019/2020

Abstract

Abstract

Contents

Chapter 1

Introduction

Brief introduction of the work

1.1 Neural Network and Deep Learning

A neural network is an interconnected structure of simple procedurals units, called nodes. Their functionality is inspired by the animals' brain and from the works on learning and neural plasticity of Donald Hebb [2]. From his book :

> *Let us assume that the persistence or repetition of a reverberatory activity (or "trace") tends to induce lasting cellular changes that add to its stability.[...] When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased*

which is an attempt to describe the change of strenght in neural relations as a consequence of stimulations. From the so-called *Hebbian Theory* rose the first computational models such as the *Perceptron*, *Neural Networks* and the modern *Deep Learning*. The development of learning-based algorithms didn't catch up with the expected results until recently, mainly due to the exponential increase in available computational resources.

From a mathematical point of view, a neural network is a composition of non-linear multi-parametric functions. During the *training phase* the model tunes its parameters, starting from random ones, by minimizing the error function (called also loss or cost). Infact, machine learning problems are just optimization problems where the solution is not given in an analytical form, therefore trough iteratives techniques (generally some kind of gradient descent) we progressively approximate the correct result.

In general, there are 3 different kind of approach to learning:

- **supervised** It exists a labeled dataset in which the relationship between features (input) and expected output is known. During training, the model is presented

with many examples and it corrects its answers based on the correct response. Some problems tied to supervised algorithms are classification, regression, object detection, segmentation and super-resolution.

- **unsupervised** In this case, a labeled dataset does not exist, only the inputs data are available. The training procedure must be tailored around the problem under study. Some examples of unsupervised algorithms are clustering, autoencoders, anomaly detection.

- **reinforced** the model interacts with a dynamic environment and tries to reach a goal (e.g. winning in a competitive game). For each iteration of the training process we assign a reward or a punishent, relatively to the progress in reaching the objective.

This work will focus on models trained using labeled samples, therefore in a supervised environment.

Perceptron

The Perceptron (also called *artificial neuron*) is the fundamental unit of every neural network and it is a simple model for a biological neuron, based on the works of Rosenblatt [4]. The *perceptron* receives N input values $x_1, x_2, ...x_N$ and the output is just a linear combination of the inputs plus a bias :

$$y = \sigma(\sum_{k=1}^{N} w_k x_k + w_0) \tag{1.1}$$

where σ is called *activation function* and $w_0, w_1, ...w_N$ are the trainable weights.

Originally, the activation function was the *Heaviside step function* whose value is zero for negative arguments and one for non-negative arguments:

$$H(x) := \begin{cases} 0 \text{ if } x < 0 \\ 1 \text{ if } x \geq 0 \end{cases} \tag{1.2}$$

In this case the perceptron is a *linear discriminator* and as such, it is able to learn an hyperplane which linearly separates two set of data. The weights are tuned during the training phase following the given update rule, usually :

$$\boldsymbol{w}_{n+1} = \boldsymbol{w}_n + \eta(t - y)\boldsymbol{x} \tag{1.3}$$

where η is the learning rate ($\eta \in [0, 1]$) and t is the true output. If the input instance is correctly classified, the error $(t-y)$ would be zero and no weight is changed. Otherwise, the hyperplane is moved towards the misclassified example. Repeating this process will lead to a convergence only if the two classes are linearly separable.

Fully Connected Structure

The direct generalization of a simple perceptron is the *Fully Connected Artificial Neural Network* (or *Multy Layer Perceptron*). It is composed by many Perceptron-like units called nodes, any one them performs the same computation as formula 1.3 and *feed* their output *forward* to the next layer of nodes. A typical representation of this type of network is shown in figure 1.1:

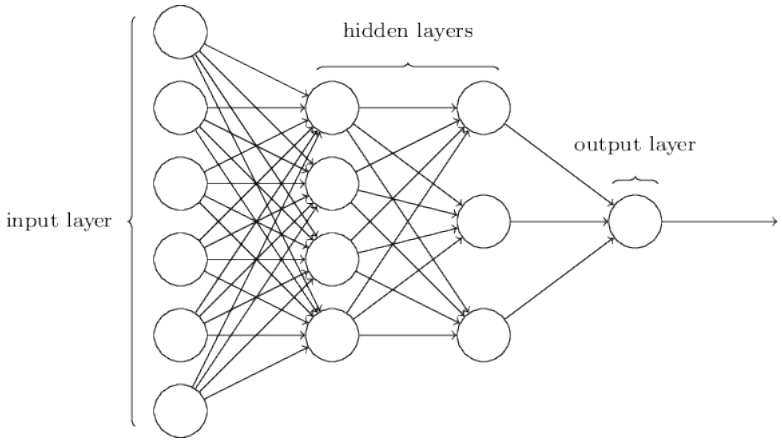

Figure 1.1: *A common representation of a neural network: a single node works as the perceptron described above.*

While the number of nodes in the input and output layers is fixed by the data under analysis, the best configuration of hidden layers is still an open problem.

The mathematical generalization from the perceptron is simple, indeed given the i-th layer its output vector $\boldsymbol{y}_i$ reads:

$$\boldsymbol{y}_i = \sigma(W_i \boldsymbol{y}_{i-1} + \boldsymbol{b}_i) \tag{1.4}$$

where W_i is the weights matrix of layer i and $\boldsymbol{b}_i$ is the i-th bias vector, equivalent to w_0 in the perceptron case. The output of the i-th layer becomes the input of the next one until the output layer yields the network's answer.

As before, σ is the activation function which can be different for every node, but it usually differs only from layer to layer. The choice of the best function for a given problem is still an open issue.

In a supervised environment, the model output is compared to the desired output (*truth*) by means of a cost function. An example of cost function is the sum of squared error :

$$C(W) = \frac{1}{N} \sum_{j=1}^{N} (y_j - t_j)^2 \tag{1.5}$$

where N is the dimensionality of the output space. C is considered as a function of the model's weights only since input data and true label t are fixed.

overcome perceptron problems

Those architectures are *universal approximators*, that means given an arbitrarly complex function, there is a fully connected neural network that can approximate it.

This type of network is called *feed forward* because the information flows linearly from the input to the output layer: however, it exists a class of models called *Recurrent* where this is not the case anymore and feedback loop are possible, but they are outside the scope of this work.

Gradient Descent

To minimize the loss function an update rule for the weights is needed. Given a cost funtion $C(w)$, the most simple one is the gradient descent:

$$w \leftarrow w - \eta \nabla_w C \tag{1.6}$$

The core idea is to modify the parameters by a small step in direction that minimize the error function. The lenght of the step is given by the *learning rate* η, which is a hyperparameter chosen by the user, while the direction of the step is given by $-\nabla_w C$, which point towards the steepest descent of the function landscape.

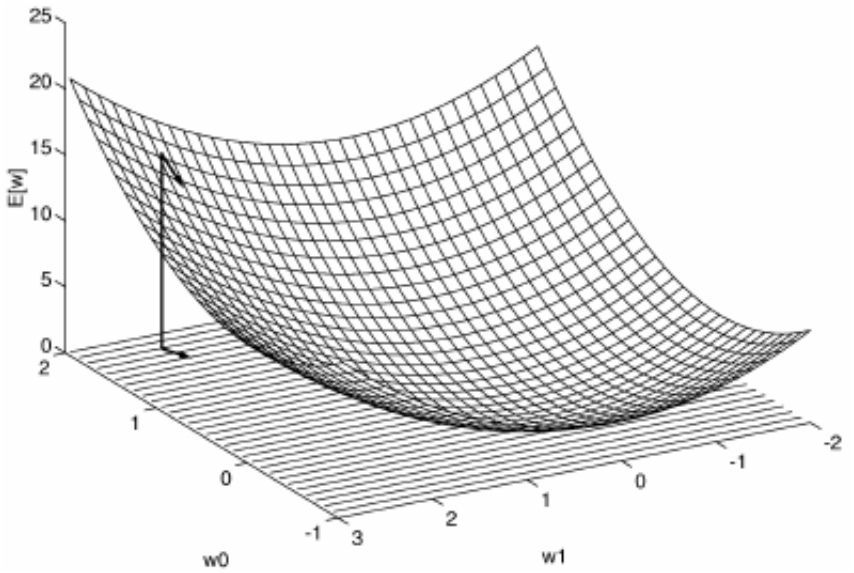

Figure 1.2: *Visual example of gradient descent for a model with 2 weights. The idea is to modify the weights to follow the direction of steepest descent for the landscape of the error function*

The speed at which the algorithm converge to a solution and the precision of said

solution are greatly influenced by the update rule. More complex and efficient update rules do exist, but they follow the same idea as the gradient descent.

Error Back Propagation

The most common algorithm used to compute the updates to weights in the learning phase is the *Error Back Propagation*. Given a differentiable cost function $C(W)$, let's define :

$$\boldsymbol{z}_l = W_l \boldsymbol{y}_{l-1} + \boldsymbol{b}_l \tag{1.7}$$

$$\boldsymbol{a}_l = \sigma(\boldsymbol{z}_l) \tag{1.8}$$

respectively the de-activated and activated output vectors of layer l for a model with L total layer, and:

$$\boldsymbol{\delta}_l = (\frac{\partial C}{\partial z_l^1}, \dots, \frac{\partial C}{\partial z_l^N}) \tag{1.9}$$

as the vector of errors of layer l, then we can write the 4 equations of back propagation for the fully connected neural network [3]:

$$\boldsymbol{\delta}_L = \nabla_a C \odot \sigma'(\boldsymbol{z}_L) \tag{1.10}$$

$$\boldsymbol{\delta}_l = (W_{l+1}^T \boldsymbol{\delta}_{l+1}) \odot \sigma'(\boldsymbol{z}_l) \tag{1.11}$$

$$\frac{\partial C}{\partial b_l^j} = \delta_l^j \tag{1.12}$$

$$\frac{\partial C}{\partial w_l^{jk}} = a_{l-1}^k \delta_l^j \tag{1.13}$$

where $\odot$ is the element-wise product.

Those equations can be generalized for others kind of layer, as I will show in the next chapters.
For a general model, the full training algorithm is :

- start with random parameters
- compute the output for one of the inputs
- computes the loss function and the gradients
- updates the parameters following the update rule.
- iterate from step 2 until the loss is sufficiently small

1.2 Super Resolution

Super resolution is class of techniques

1.3 Magnetic Resonance

Chapter 2

Algorithms

2.1 Byron

2.2 Convolutional Neural Network

A Convolutional Neural Network (CNN) is a specialized kind of neural network for processing data that has known grid-like topology [1]. The name indicates that one of the functions employed by the network is a convolution. In a continuos domain is defined as:

$$(f * g)(t) = \int_{-\infty}^{+\infty} f(\tau)g(t-\tau)d\tau \tag{2.1}$$

The first function f is usually referred to as the input and the second function g as kernel. For Image Processing applications we can define a 2-dimensional discrete version of the convolution in a finite domain using an image I as input and a 2 dimensional kernel k:

$$C[i,j] = \sum_{u=-N}^{N} \sum_{v=-M}^{M} k[u,v] \cdot I[i-u, j-v] \tag{2.2}$$

where $C[i,j]$ is the pixel value of the output image and N, M are the kernel dimensions.

2.3 Layers

Convolutional Layer

Pixel Shuffle Layer

Connected Layer

Loss Function

2.4 Timing

2.5

Chapter 3

Dataset and Methodology

3.1 Dataset

3.2 Models

Chapter 4

Results

Bibliography

[1] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[2] D. Hebb. *The Organization of Behavior*. 1949.

[3] M. A. Nielsen. *Neural Network and Deep Learning*. Determination Press, 2015.

[4] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, pages 65–386, 1958.