

A General Feedforward Neural Network Model

Cédric GEGOUT^{•♦*}, Bernard GIRAU[•] & Fabrice ROSSI^{1♦*}

★ THOMSON-CSF/SDC/DPR/R4,
7 rue des Mathurins, 92223 BAGNEUX FRANCE
Phone: (33 1) 40 84 29 05 Fax: (33 1) 40 84 29 50

◇ Ecole Normale Supérieure de Paris,
45 rue d'Ulm, 75005 PARIS FRANCE

● Ecole Normale Supérieure de Lyon,
Laboratoire de l'Informatique du Parallélisme,
46 avenue d'Italie, 69007 LYON FRANCE

e-mail: gegout@clipper.ens.fr, bgirau@lip.ens-lyon.fr et rossi@clipper.ens.fr
WWW URL: <http://acacia.ens.fr:8080/home/rossi/index.uk.html>
<http://www.ens-lyon.fr/~bgirau/>

May 1995

NeuroCOLT Technical Report Series

NC-TR-95-041²

May 1995

Produced as part of the ESPRIT Working Group in Neural and
Computational Learning, NeuroCOLT 8556

¹Up to date contact informations for Fabrice Rossi are available at <http://apiacoa.org/>

²Available at <http://apiacoa.org/publications/1995/neurocolt1995.pdf>

Abstract

In this paper, we generalize a model proposed by Léon Bottou and Patrick Gallinari in [3]. This model gives a general mathematical description of feedforward neural networks, for which standard models, such as Multi-Layer Perceptrons or Radial Basis Function based neural networks, are only particular cases. A generalized back-propagation, which gives an efficient way to compute the differential of the function computed by the neural network, is introduced and carefully proved. We also introduce an evaluation of the theoretical time needed to compute the differential with the help of both direct algorithm and back-propagation.

Contents

1	Introduction	4
2	The neuron	5
2.1	A formal neuron	5
2.2	Multiple inputs	5
3	Graph	6
3.1	A simple oriented graph	6
3.2	Ordered oriented graph	7
3.3	Some notations	7
4	A general neural network	8
4.1	The model	8
4.2	Additional definitions	9
5	Computing with the model	9
5.1	Intuitive point of view	10
5.2	Mathematical point of view	10
6	Computing the differentials	11
6.1	Simple property	12
6.2	Direct method	12
7	Back-propagation	13
7.1	“Free” neural networks	13
7.2	Additional notations	16
7.3	The main result	17
7.4	Output functions	22
8	Input Sharing	24
9	Complexity	24
9.1	Notations and preliminary remarks	25
9.2	Direct algorithm	25
9.3	Back-propagation	26
9.4	Application	27

10 Some examples	27
10.1 Neuron-based Multi-Layer Perceptron	27
10.2 Layer-based Multi-Layer Perceptron	31
11 Conclusion	33

1 Introduction

A huge amount of experimental works has proved that multi-layer perceptrons (MLP) are well suited for classification tasks, for which the goal is to separate vectorial inputs into several classes. It is also well known that MLPs are universal approximators (see [1, 5, 7]): given an arbitrary continuous mapping from a compact set (subset of \mathbb{R}^n) into \mathbb{R}^p , and an arbitrary accuracy, there exists a two layer perceptron that approximates the given function with the given accuracy. Two main problems are the design of such a MLP with a minimum number of neurons, and the training of a given MLP so as to minimize the approximation error with respect to a given function. Theoretical results define some methods to answer the first question, but they build still too huge neural networks. Experimental results seem to prove that the second problem is more difficult than the classification problem, especially to obtain the *best* approximation (it is due to the well known problem of local minima).

The MLP model has been modified in many different ways to simplify both designing and training processes. In a MLP, a neuron collects outputs from the previous layer, it multiplies each output by a synaptic weight, it sums the resulting values and then it applies a transfer function to the sum. The simplest modification idea is to change the transfer function, using a sinus [4] or a gaussian function [8], for instance, instead of the standard sigmoid function. It is also possible to modify the preprocessing computation. A vectorial threshold may be applied to the output of the previous layer before the weighting process ([13]). The distance between the output of the previous layer and a prototype vector may be computed before using a transfer function. This idea was developed to use radial basis functions as transfer functions ([9, 10, 11]). A non linear process (such as a quadratic form, see [12]) may be used instead of the linear process (i.e. the weighted sum). Some authors have also proposed to use multidimensional wavelets as neurons (see for instance [14]). This method strongly modifies the neuron structure, when compared to the standard MLP neuron.

All these models are universal approximators. Choosing the best one is difficult, since there is no theoretical result to compare their performance. Indeed a particular model might be well suited for a particular application and unadapted to another one. Despite their differences, these models share a common principle: they use an acyclic graph of simple units to compute a complex parametric vectorial function. This general point of view can be translated into a mathematical model which generalizes the notion of feedforward neural network. This method was proposed by Léon Bottou and Patrick Gallinari in [3]. Their key idea was to allow the cooperation of modules of different kinds. In this paper, we extend the proposed model to describe a general mathematical model for feedforward neural networks. This model is able to handle many models, among which all the models derived from the standard MLP model.

Following Bottou and Gallinari, we introduce a generalized back-propagation algorithm which allows us to compute the gradient of the error made by a neural network in an efficient way. We extend the algorithm proposed by Bottou and Gallinari to compute the differentials of the function computed by the neural network too. It allows us to view a neural network itself as a neuron. Moreover, we give a very precise proof of this back-propagation and therefore we clarify some imprecise aspects in Bottou and Gallinari's proofs.

The differentials of the function computed by the neural network can also be computed with the help of a direct algorithm. In order to compare both methods, we compute the theoretical amount of operations needed by both algorithms. Our work shows that these amounts can not be directly compared in the general case. A study of some particular cases shows that the

back-propagation is not always the best algorithm for arbitrary feedforward neural networks.

2 The neuron

2.1 A formal neuron

Intuitively, a formal neuron is a simple computational unit which transforms an **input** into an **output**. In the MLP model, connection weights are used so as to preprocess the input of a neuron. The connection weights may be included in the neuron itself, so that the whole computation (preprocessing and function application) is performed within the neuron. The connection weights are then called parameters, which modify the relationship between the input and the output of the neuron. In a more general way, a **neuron** is described by three vectorial spaces and a function. The vectorial spaces are the input space (to which the input vector belongs), the parameter space (the parameter vector generalizes the notion of connection weights) and the output space. The function describes the relationship between the output of the neuron and its input and parameter vectors. More formally:

Definition 1 *Let I , W and O be three vectorial spaces on \mathbb{R} of finite dimensions.*

*A **neuron** is a function from $I \times W$ into O .*

*A **differentiable neuron** is a differentiable function from $I \times W$ into O . If N is such a function, then dN_i is the partial differential of N with respect to its first input (its second input is considered to be constant) and dN_w is the partial differential of N with respect to its second input (constant first input).*

In this definition, I is the input space of the neuron and W is its parameter space (or the weight space to be close to the standard MLP model). The following example describes a standard MLP neuron with respect to the previous definition:

Example 1 *Let $(w_j)_{j=1..n}$ be the connection weights of a MLP neuron and let t be its threshold. If the output of the previous layer of the neural network (possibly the input of the neural network) is $(x_j)_{j=1..n}$ then the output of the neuron is $f(\sum_{j=1}^n w_j x_j + t)$, where f is the transfer function of the neuron. Let I , W and O be respectively \mathbb{R}^n , \mathbb{R}^{n+1} and \mathbb{R} . Let N be the function from $I \times W$ to O defined by:*

$$N((i_1, \dots, i_n), (p_1, \dots, p_{n+1})) = f\left(\sum_{j=1}^n p_j i_j + p_{n+1}\right)$$

If $p = (w_1, \dots, w_n, t)$ and $i = (x_1, \dots, x_n)$, then $N(i, p)$ is exactly the output of the MLP neuron. This neuron is therefore a mathematical description of the standard MLP neuron.

2.2 Multiple inputs

Since the input vector of a neuron belongs to an arbitrary vectorial space, any number of real inputs can be handled. But as the output of several neurons will be connected to the input of one neuron, it is useful to describe multiple input neurons:

Definition 2 *Let n be a positive integer and let I_1, \dots, I_n , W and O be vectorial spaces on \mathbb{R} of finite dimensions.*

A **(differentiable) n -input neuron** is a (differentiable) function from $I_1 \times \dots \times I_n \times W$ into O .

If N is a differentiable n -input neuron, dN_{i_j} is the partial differential of N with respect to its j -th input, considering all other inputs and the parameter to be constant.

Spaces of same dimension are isomorphic. For instance, $\mathbb{R}^{p+q} \simeq \mathbb{R}^p \times \mathbb{R}^q$. The vector (a, b, c, d) is in \mathbb{R}^4 , or in $\mathbb{R} \times \mathbb{R}^3$ if it is written as $((a), (b, c, d))$, or in $\mathbb{R}^2 \times \mathbb{R}^2$ if it is written as $((a, b), (c, d))$. In our model, we will always write \mathbb{R}^k for the vectorial spaces, and we will always identify spaces of same dimension with the help of this standard isomorphism. Therefore, a n -input neuron is only a convenient way to describe a 1-input neuron.

The following example describes the standard MLP neuron with respect to this second definition:

Example 2 We use here the same notations as in example 1.

Let W and O be respectively \mathbb{R}^{n+1} and \mathbb{R} . Let $I_j = \mathbb{R}$ for j in $\{1, \dots, n\}$. Let N be the function from $I_1 \times I_2 \times \dots \times I_n \times W$ into O defined by:

$$N(i_1, \dots, i_n, (p_1, \dots, p_{n+1})) = f \left(\sum_{j=1}^n p_j i_j + p_{n+1} \right)$$

If $p = (w_1, \dots, w_n, t)$ and $i_1 = (x_1)$, $i_2 = (x_2), \dots, i_n = (x_n)$, then $N(i, p)$ is exactly the output of the MLP neuron. This neuron is therefore a mathematical description of the standard MLP neuron.

This example shows that the model is very flexible and allows several descriptions of a single object.

3 Graph

The easiest way to describe arbitrary feedforward neural networks is to use *graphs*. This section gives an introduction to this notion.

3.1 A simple oriented graph

Intuitively, a graph is a set of nodes with connections between them. We choose the following mathematical definitions:

Definition 3 An oriented graph \mathcal{G} is a pair $(\mathcal{N}, \mathcal{E})$ of sets which fulfils the following conditions:

- \mathcal{N} is a **finite** set of nodes ;
- \mathcal{E} is a subset of \mathcal{N}^∞ , and $e = (x, y) \in \mathcal{E}$ is an **edge** of the graph if there is a connection from x to y .

Definition 4 Let $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ be an oriented graph and let $n \in \mathcal{N}$ be a node of \mathcal{G} .

1. $\text{Pred}(n) = \{p \in \mathcal{N} \mid (\sqrt{}, \backslash) \in \mathcal{E}\}$ is the set of the nodes that have a connection to n . These nodes are the **predecessors** of n .

2. $Succ(n) = \{p \in \mathcal{N} \mid (\setminus, \surd) \in \mathcal{E}\}$ is the set of the nodes that receive a connection from n . These nodes are the **successors** of n .

It is obvious that the architecture of a standard neural network such as the MLP can be described thanks to the notion of oriented graph. Each neuron is a node of the graph.

Any feedforward structure can be implemented thanks to a non-cyclic graph. The inputs of a neuron (as defined in section 2) are the outputs of its predecessors.

A problem is to put an order on the inputs. For instance, let a be a neuron which predecessor set is $Pred(a) = \{b, c\}$. Let us assume that a is a 2-input neuron and let b' and c' be respectively the output of neurons b and c . If p is a parameter vector for a , the output of a might be $a(b', c', p)$ as well as $a(c', b', p)$. In order to decide which node is connected to which *input* of another node, information must be added to the standard oriented graph model. This is the subject of the following subsection.

3.2 Ordered oriented graph

Definition 5 An ordered oriented graph \mathcal{G} is a triple $(\mathcal{N}, \mathcal{E}, <)$, which fulfils the following conditions:

- \mathcal{N} is a **finite totally ordered** set of nodes. It may be considered as a sequence of nodes N^1, N^2, \dots ;
- \mathcal{E} is a subset of \mathcal{N}^∞ , and $e = (x, y) \in \mathcal{E}$ is an **edge** of the graph if there is a connection from x to y ;
- $<$ is a function from \mathcal{N} into $\mathcal{P}(\mathcal{N}^\infty)$. It associates to each node N^i in \mathcal{N} a total order $<_{N^i}$ on its predecessor set, $Pred(N^i)$. This set is therefore a sequence and it can be referred to $Pred(N^i)_k$ for its k -th element.

3.3 Some notations

In order to simplify the remainder of the paper, some notations and some general assumptions must be introduced:

- $\mathcal{G} = (\mathcal{N}, \mathcal{E}, <)$ is an ordered graph with exactly n nodes,
- N^1, \dots, N^n is the sequence of the graph nodes,
- $P(N^k) = P(k)$ is the set of the predecessors of N^k ,
- $S(N^k) = S(k)$ is the set of the successors of N^k ,
- Node N^k has exactly p^k predecessors and s^k successors,
- $P(k)_1, \dots, P(k)_{p^k}$ is the sequence of the predecessors of N^k ,
- $S(k)_1, \dots, S(k)_{s^k}$ is the sequence of the successors of N^k .

In general, superscripts correspond to node indices and subscripts correspond to input or output indices. It may be noted that the ordering of the successor sets are arbitrary.

The notion of predecessor may be generalized for an arbitrary node N :

- $P^0(N) = \{N\}$;
- $P^k(N) = \{M \in \mathcal{N} \mid \exists \mathcal{Q} \in \mathcal{P}^{\parallel-\infty}(\mathcal{N}) \text{ so that } (M, \mathcal{Q}) \in \mathcal{E}\}$ (i.e. $P^1(N) = P(N)$) ;
- $P^+(N) = \bigcup_{k=1}^{\infty} P^k(N)$;
- $P^*(N) = P^0(N) \cup P^+(N)$.

Similar sets can be defined in order to generalize the notion of successor.

- $S^0(N) = \{N\}$;
- $S^k(N) = \{M \in \mathcal{N} \mid \exists \mathcal{Q} \in \mathcal{S}^{\parallel-\infty}(\mathcal{N}) \text{ so that } (\mathcal{Q}, M) \in \mathcal{E}\}$;
- $S^+(N) = \bigcup_{k=1}^{\infty} S^k(N)$;
- $S^*(N) = S^0(N) \cup S^+(N)$.

4 A general neural network

4.1 The model

The main idea has been presented at the end of subsection 3.1. Let us now give a mathematical definition:

Definition 6 *A feedforward neural network is an ordered oriented graph $\mathcal{G} = (\mathcal{N}, \mathcal{E}, <)$ which fulfils the following conditions:*

1. *The graph has **no cycle**.*
2. *The elements of \mathcal{N} are **k-input neurons** (k depends on the neuron). The output space of N^k is O^k and its parameter space is W^k .*
3. *If $p^k > 0$ then neuron N^k is a p^k -input neuron (it means that if a neuron has got predecessors, it has exactly one input for each of its predecessors in the graph). The input spaces of N^k are $I_1^k, \dots, I_{p^k}^k$.*
4. *If $p^k = 0$ then neuron N^k is a 1-input neuron with input space I^k .*
5. *If $p^k > 0$ then the following condition holds for each input i of the neuron N^k : $\dim I_i^k = \dim O^{P(k)_i}$.*

These conditions can be expressed from an intuitive point of view:

1. The first condition avoids recurrent architectures.
2. The second condition gives the neural aspect to the neural network.
3. The third condition introduces a binding between the inputs of a neuron and its predecessors in the graph. Of course the outputs of its predecessors are used as its inputs to compute its output.

4. The fourth condition is a simple convention. If a neuron has no predecessor, it is connected to the *outside*. We assume here a single input (of course, this input is a vector, and therefore this assumption does not introduce any restriction).
5. The fifth condition is a technical one and allows to use the output of the i -th predecessor of the neuron N^k as its i -th input.

In the remainder of the paper, we will identify each neuron with its rank in the node set. For instance, let us assume that $P(N^3) = \{N^2, N^1\}$, i.e., $P(3)_1 = P(N^3)_1 = N^2$ and $P(3)_2 = N^1$. To study the output space of the first predecessor of N^3 , we should write O^2 . But from a strict mathematical point of view, $O^{P(k)_i}$ has no meaning in the general case. Therefore a function *rank* from \mathcal{N} into $\{1, \dots, n\}$ must be defined. This function gives the rank of a node in \mathcal{N} . Nevertheless, writing $O^{\text{rank}(P(k)_i)}$ is rather cumbersome, so that we will omit the *rank* function in the remainder of the paper.

4.2 Additional definitions

Let $\mathcal{G} = (\mathcal{N}, \mathcal{E}, <)$ be a feedforward neural network.

- In is the **subset of \mathcal{N} which elements have no predecessor**, i.e., $In = \{N \in \mathcal{N} \mid \mathcal{P}(N) = \emptyset\}$. In has in elements. It is a totally ordered set (order induced by the order on \mathcal{N}) and In_1, \dots, In_{in} is the ordered sequence of its elements. The elements of In are connected to the “outside” by means of their inputs.
- Out is the **subset of \mathcal{N} which elements have no successor**, i.e., $Out = \{N \in \mathcal{N} \mid \mathcal{S}(N) = \emptyset\}$. Out has out elements and is totally ordered. Out_1, \dots, Out_{out} is the ordered sequence of its elements. The elements of Out are connected to the “outside” by means of their outputs.
- The vectorial space $I = \prod_{k=1}^{in} I^{In_k}$ is **the input space** of the neural network. I is the product of the input spaces of the neurons that belong to In .
- The vectorial space $O = \prod_{k=1}^{out} O^{Out_k}$ is **the output space** of the neural network. O is the product of the output spaces of the neurons that belong to Out .
- The vectorial space $W = \prod_{k=1}^n W^k$ is **the parameter space** of the neural network. W is the product of the parameter spaces of the network neurons.

Thanks to these vectorial spaces, a neural network may be considered as a *neuron*. The following section explains how to define the function computed by the neural network.

5 Computing with the model

Let $\mathcal{G} = (\mathcal{N}, \mathcal{E}, <)$ be a feedforward neural network. Let $x = (x_1, \dots, x_{in})$ be a vector of the input space I of the neural network, and let $w = (w^1, \dots, w^n)$ be a vector of the parameter space W of the neural network. To define the output $G(x, w)$ of the neural network for the input x and the parameter w , an output is computed recursively for each neuron in the neural network. The output of N^k is called o^k .

5.1 Intuitive point of view

1. Let N^l be an *input* node, i.e., $N^l \in In$. Then N^l may be written $N^l = In_k$ as the k -th element of In . The output of N^l is then defined as $o^l = N^l(x_k, w^l)$. **The input neurons receive their inputs from the outside.**
2. Let N^l be a non-input neuron and let us assume that the output of each predecessor of N^l has been computed. Then the output of N^l is defined by $o^l = N^l(o^{P(l)_1}, o^{P(l)_2}, \dots, o^{P(l)_{p^l}}, w^l)$. **The “inside” neurons receive their inputs from their predecessors.**
3. Let us now assume that the output of every neuron in the network has been computed. The output of the neural network is $G(x, w) = (o^{Out_1}, \dots, o^{Out_{out}})$. **The output of the neural network is obtained thanks to the outputs of the output neurons.**

5.2 Mathematical point of view

To prove that the computation method that has been presented in the previous subsection is correct, a computation order must be found to ensure that the condition of point 2 is fulfilled at each step of the algorithm. This order is defined with the help of the notion of **layer**.

Property 1 *Let $\mathcal{G} = (\mathcal{N}, \mathcal{E}, <)$ be a feedforward neural network. There exists an unique positive integer $l(\mathcal{G})$ and an unique partition of the node set \mathcal{N} defined by the l subsets $L^1, \dots, L^{l(\mathcal{G})}$, which satisfy the following conditions:*

1. $L^1 = In$;
2. $\forall k > 1, L^k = \left\{ N \in \mathcal{N} \mid \mathcal{P}(\mathcal{N}) \subset \bigcup_{l=1}^{k-1} L^l \text{ and } \exists N' \in \mathcal{P}(\mathcal{N}), N' \in L^{k-1} \right\}$.

The L^i are the **layers** of the neural network and $l(\mathcal{G})$ is the **number of layers** of the neural network.

Proof: Properties 1 and 2 allow a recursive definition of the partition. Let us show that this definition is valid:

- L^1 is defined. In order to have a partition, the condition $L^1 \neq \emptyset$ must be fulfilled. The graph is not cyclic, which implies $In \neq \emptyset$ (If any neuron has got predecessors, an infinite sequence N^{i_1}, N^{i_2}, \dots can be built so that for all j , N^{i_j} is a predecessor of $N^{i_{j+1}}$. Since \mathcal{N} is finite, this sequence contains a cycle.)
- Let us now assume that $k \geq 1$ and that L^1, \dots, L^k have been defined so that:
 1. $\forall i \leq k, L^i \subset \mathcal{N}$;
 2. $\forall i \leq k, \forall j \leq k, (i \neq j) \Rightarrow L^i \cap L^j = \emptyset$;
 3. $\forall i \leq k, i > 1 \Rightarrow L^i = \left\{ N \in \mathcal{N} \mid \mathcal{P}(\mathcal{N}) \subset \bigcup_{l=1}^{i-1} L^l \text{ and } \mathcal{P}(\mathcal{N}) \cap L^{i-1} \neq \emptyset \right\}$.
 4. $\forall i \leq k, L^i \neq \emptyset$;

Let now L^{k+1} be $\left\{ N \in \mathcal{N} \mid \mathcal{P}(\mathcal{N}) \subset \bigcup_{l=1}^k L^l \text{ and } \mathcal{P}(\mathcal{N}) \cap L^k \neq \emptyset \right\}$.

1. Of course $L^{k+1} \subset \mathcal{N}$;
2. If $j \leq k$ and $N^l \in L^j$, then $\mathcal{P}(N^l) \cap L^k = \emptyset$. Therefore $\forall j \leq k, L^j \cap L^{k+1} = \emptyset$.
3. By definition, $\forall i \leq k+1, i > 1 \Rightarrow L^i = \left\{ N \in \mathcal{N} \mid \mathcal{P}(\mathcal{N}) \subset \bigcup_{l=1}^{i-1} L^l \text{ and } \mathcal{P}(\mathcal{N}) \cap L^{i-1} \neq \emptyset \right\}$.
4. (a) If $L^{k+1} \neq \emptyset$, the second hypothesis is satisfied.

- (b) If $L^{k+1} = \emptyset$ and if $R = \mathcal{N} - \bigcup_{l=1}^k L^l$ satisfies $R \neq \emptyset$, let us build a sequence of distinct nodes $r^p \in R$ such that $r^{p+1} \in P(r^p)$. Let first r^0 be in R . Let us assume that the sequence is built from 0 to p . As $r^p \notin L^1$, $P(r^p) \neq \emptyset$. If for all $r \in P(r^p)$, $r \notin R$, then $P(r^p) \subset \bigcup_{j=1}^k L^j$, which implies $r^p \in L^q$ for a particular q with $1 < q \leq k$. This is inconsistent with $r^p \in R$. Therefore there exists $r^{p+1} \in P(r^p) \cap R$. Since the graph is not cyclic and since \mathcal{N} is finite, the sequence $\{r^p\}$ can not exist. Therefore, if $L^{k+1} = \emptyset$, then $\mathcal{N} = \bigcup_{l=1}^k L^l$.
- $\forall j \leq k$, $L^j \neq \emptyset$ implies $|\bigcup_{j=1}^k L^j| \geq k$. As \mathcal{N} is finite, there exists a particular k , such that $L^{k+1} = \emptyset$. Therefore, this recursive building is finite. Then L^1, \dots, L^k is the required partition and $l(\mathcal{G}) = \parallel$.

The obtained partition is unique because of condition 2. ■

Intuitively, the first layer consists of the input neurons. The second layer consists of the neurons which are not in the first layer and which have all their predecessors in the first layer. Any other layer consists of the neurons which are not in the previous layers and which have at least one predecessor in the preceding layer.

With the help of the order defined by the layers, it is now obvious that the intuitive definition of the output of the neural network is correct as long as the neuron outputs are computed in a *feedforward* way with respect to the layers.

Let us now give a formal definition of $o^l(x, w)$.

Definition 7 Let $\mathcal{G} = (\mathcal{N}, \mathcal{E}, <)$ be a feedforward neural network. Let $x = (x_1, \dots, x_{in}) \in I$ be an input vector and let $w = (w^1, \dots, w^n) \in W$ be a parameter vector. For each l , $1 \leq l \leq n$, $o^l(x, w)$ is defined with the help of the following stepwise construction:

- 1 For $N^l \in L^1 = In$, $N^l = In_k$. Then $o^l(x, w) = N^l(x_k, w^l)$;
- 2 For $N^l \in L^2$, $o^l(x, w) = N^l(o^{P(l)_1}(x, w), \dots, o^{P(l)_{p^l}}(x, w), w^l)$. This definition is correct, since $N^l \in L^2 \Rightarrow P(l) \subset L^1$ and therefore every $o^{P(l)_k}(x, w)$ has already been defined ;
- ...
- k For $N^l \in L^k$, $o^l(x, w) = N^l(o^{P(l)_1}(x, w), \dots, o^{P(l)_{p^l}}(x, w), w^l)$. This definition is correct, since $N^l \in L^k \Rightarrow P(l) \subset \bigcup_{j=1}^{k-1} L^j$ and therefore every $o^{P(l)_k}(x, w)$ has already been defined ;
- ...

Finally, $G(x, w)$ is obtained by $G(x, w) = (o^{Out_1}(x, w), \dots, o^{Out_{out}}(x, w))$

This definition allows to consider a neural network itself as a neuron.

6 Computing the differentials

If we use differentiable neurons, the function (neuron) computed by the neural network is also differentiable, since it is a composite function. The main problem is to compute efficiently the differentials of this function. In this section, we introduce the direct algorithm which is simply the application of the chain rule. In the following sections, we always assume that the neurons are differentiable.

6.1 Simple property

As explained before, the output of a neuron in the neural network is considered as a function of x and w , $o^l(x, w)$. But o^l does not depend on parameters or inputs coming from nodes which do not belong to $P^*(l)$. More formally:

Property 2 *Let $\mathcal{G} = (\mathcal{N}, \mathcal{E}, <)$ be a feedforward neural network. Let N^l be a neuron of \mathcal{G} . Then for an arbitrary input vector x and for an arbitrary parameter vector w :*

$$\forall N^k \notin P^*(l), \quad \frac{\partial o^l}{\partial w^k}(x, w) = 0, \quad (1)$$

and,

$$\forall N^k = In_j \in In, \quad N^k \notin P^*(l) \Rightarrow \frac{\partial o^l}{\partial x_j}(x, w) = 0 \quad (2)$$

Proof: For the input nodes, the property is obviously satisfied. Let us assume it is satisfied for the nodes that belong to layer L^q with $q \leq p$, and let N^l be a node of layer L^{p+1} .

If the chain rule is applied to the definition of $o^l(x, w)$, the following formula is obtained (as $k \neq l$)

$$\frac{\partial o^l}{\partial w^k}(x, w) = \sum_{j=1}^{p_l} dN_{i_j}^l \left(o^{P(l)_1}(x, w), \dots, o^{P(l)_{p_l}}(x, w), w^l \right) \frac{\partial o^{P(l)_i}}{\partial w^k}(x, w)$$

$N^k \notin P^*(l) \Rightarrow \forall N \in P(l), N^k \notin P^*(N)$. Since $N \in P(l) \Rightarrow N \in \bigcup_{q=1}^p L^q$, each $N \in P(l)$ satisfies the property, and the required conclusion is obtained.

A similar proof can be used for the second property. ■

6.2 Direct method

Theorem 1 *Let $\mathcal{G} = (\mathcal{N}, \mathcal{E}, <)$ be a feedforward neural network. Let N^l be an arbitrary node. Let x and w be arbitrary input and parameter vectors. Then:*

- if $N^l = In_k$:

- if $j \neq k$:

$$\frac{\partial o^l}{\partial x_j}(x, w) = 0 \quad (3)$$

- if $j = l$:

$$\frac{\partial o^l}{\partial w^j}(x, w) = 0 \quad (4)$$

- and:

$$\frac{\partial o^l}{\partial x_k}(x, w) = dN_i^l(x_k, w^l) \quad (5)$$

$$\frac{\partial o^l}{\partial w^l}(x, w) = dN_w^l(x_k, w^l) \quad (6)$$

- if $N^l \notin In$:

– for all j , $1 \leq j \leq in$:

$$\frac{\partial o^l}{\partial x_j}(x, w) = \sum_{k=1}^{p^l} dN_{i_k}^l \left(o^{P(l)_1}(x, w), \dots, o^{P(l)_{p^l}}(x, w), w^l \right) \frac{\partial o^{P(l)_k}}{\partial x_j}(x, w) \quad (7)$$

– if $j \neq l$:

$$\frac{\partial o^l}{\partial w^j}(x, w) = \sum_{k=1}^{p^l} dN_{i_k}^l \left(o^{P(l)_1}(x, w), \dots, o^{P(l)_{p^l}}(x, w), w^l \right) \frac{\partial o^{P(l)_k}}{\partial w^j}(x, w) \quad (8)$$

– and:

$$\frac{\partial o^l}{\partial w^l}(x, w) = dN_w^l(x_k, w^l) \quad (9)$$

Proof: This is a direct application of the chain rule for composite functions. ■

7 Back-propagation

7.1 “Free” neural networks

The *global* point of view is to consider the output of a neuron in the neural network as a function $o^l(x, w)$ of the input and parameter of the neural network. The *local* point of view considers a local computation, based on local information:

$$o^l(x, w) = N^l(o^{P(l)_1}(x, w), o^{P(l)_2}(x, w), \dots, o^{P(l)_{p^l}}(x, w), w^l)$$

This local functional link is used in the direct algorithm to compute the global differential.

Let us describe now the notion of *free* network. In such a network a neuron is “removed”. Indeed a variable is added to the function computed by the neural network and this variable stands for the output of one neuron (the removed neuron) in the network. To set “free” the network with respect to the neuron N^k , a new variable f^k is introduced. It replaces the output o^k of N^k in the definition of the output of the other neurons. New functions $o^{l \rightarrow k}(x, w, f^k)$ are thus obtained. Let us give a formal definition of $o^{l \rightarrow k}$.

Definition 8 Let $\mathcal{G} = (\mathcal{N}, \mathcal{E}, <)$ be a feedforward neural network. Let $x = (x_1, \dots, x_{in}) \in I$ be an input vector and let $w = (w^1, \dots, w^n) \in W$ be a parameter vector. Let N^k be an arbitrary node. We define $o^{k \rightarrow k}(x, w, f^k) = f^k$. For each l , $1 \leq l \leq n$, and $l \neq k$, $o^{l \rightarrow k}(x, w, f^k)$ is defined with the help of the following stepwise construction:

- 1 If $N^l \in L^1 = In$, then $N^l = In_q$ and we define $o^{l \rightarrow k}(x, w, f^k) = N^l(x_q, w^l)$;
- 2 If $N^l \in L^2$, then $o^{l \rightarrow k}(x, w, f^k) = N^l(o^{P(l)_1 \rightarrow k}(x, w, f^k), \dots, o^{P(l)_{p^l} \rightarrow k}(x, w, f^k), w^l)$. This definition is correct, since $N^l \in L^2$ implies $P(l) \subset L^1$, so that each $o^{P(l)_k \rightarrow k}(x, w, f^k)$ has already been defined ;
- ...
- p If $N^l \in L^p$, then $o^{l \rightarrow k}(x, w, f^k) = N^l(o^{P(l)_1 \rightarrow k}(x, w, f^k), \dots, o^{P(l)_{p^l} \rightarrow k}(x, w, f^k), w^l)$. This definition is correct, since $N^l \in L^p$ implies $P(l) \subset \bigcup_{j=1}^{p-1} L^j$, so that each $o^{P(l)_k \rightarrow k}(x, w, f^k)$ has already been defined ;

...

Finally, let $G^k(x, w, f^k)$ be $(o^{Out_1 \rightarrow k}(x, w, f^k), \dots, o^{Out_{out} \rightarrow k}(x, w, f^k))$.

This definition is close to the definition of G . The main difference is that the recursive definition is not used so as to compute $o^{k \rightarrow k}$.

Let us first state an important (though intuitive) property:

Property 3 Let $\mathcal{G} = (\mathcal{N}, \mathcal{E}, <)$ be a feedforward neural network. Let N^k be a network node. The following conditions hold for an arbitrary input vector x , an arbitrary parameter vector w , and an arbitrary free output vector f^k :

$$\forall l \neq k \quad o^{l \rightarrow k}(x, w, o^k(x, w)) = o^l(x, w) \quad (10)$$

$$\forall N^l \notin S^*(k) \quad o^{l \rightarrow k}(x, w, f^k) = o^l(x, w) \quad (11)$$

Proof: The first equality is an obvious consequence of definition 8. The second equality can be obtained thanks to a recursive proof with respect to the layer of node N^l . The key idea is that if $N^l \notin S^*(k)$ then for each $N \in P(l)$, $N \notin S^*(k)$. Since the building of the recursion is similar to previous proofs, we omit here its details. ■

When computing the derivative of:

$$o^{l \rightarrow k}(x, w, o^k(x, w)) = o^l(x, w), \quad (12)$$

the following formula is obtained:

$$\frac{\partial o^l}{\partial w^k}(x, w) = \frac{\partial o^{l \rightarrow k}}{\partial w^k}(x, w, o^k(x, w)) + \frac{\partial o^{l \rightarrow k}}{\partial o^k}(x, w, o^k(x, w)) \frac{\partial o^k}{\partial w^k}(x, w) \quad (13)$$

$\frac{\partial o^{l \rightarrow k}}{\partial o^k}$ is a simplified notation for the partial differential of $o^{l \rightarrow k}$ with respect to the variable f^k (i.e., the free neuron).

An important result can be proved to simplify the previous formula:

Property 4 Let $\mathcal{G} = (\mathcal{N}, \mathcal{E}, <)$ be a feedforward neural network. Let N^l and N^k be two arbitrary nodes. Let x , w and f^k be arbitrary input, parameter and free output vectors. Then:

$$\frac{\partial o^{l \rightarrow k}}{\partial w^k}(x, w, f^k) = 0_{W^k, O^l} \quad (14)$$

where $0_{A,B}$ is the zero function from space A into space B .

Intuitively, this equation implies that the output of the network depends on a parameter vector w^k only through the output of the corresponding neuron N^k .

Proof: This result is obvious as w^k never appears in the recursive definition of $o^{l \rightarrow k}$, see definition 8.

More formally, we can prove it thanks to a recursion, similarly to property 2. ■

An important consequence is the following theorem:

Theorem 2 Let $\mathcal{G} = (\mathcal{N}, \mathcal{E}, <)$ be a feedforward neural network. Let N^l and N^k be two arbitrary nodes. Let x and w be arbitrary input and parameter vectors. Then:

- if $N^k \notin In$:

$$\begin{aligned} \frac{\partial o^l}{\partial w^k}(x, w) &= \frac{\partial o^{l \rightarrow k}}{\partial o^k}(x, w, o^k(x, w)) \frac{\partial o^k}{\partial w^k}(x, w) \\ &= \frac{\partial o^{l \rightarrow k}}{\partial o^k}(x, w, o^k(x, w)) dN_w^k(o^{P(k)_1}(x, w), \dots, o^{P(k)_p}(x, w), w^k) \end{aligned} \quad (15)$$

- if $N^k = In_i$:

$$\begin{aligned} \frac{\partial o^l}{\partial w^k}(x, w) &= \frac{\partial o^{l \rightarrow k}}{\partial o^k}(x, w, o^k(x, w)) \frac{\partial o^k}{\partial w^k}(x, w) \\ &= \frac{\partial o^{l \rightarrow k}}{\partial o^k}(x, w, o^k(x, w)) dN_w^k(x_i, w^k) \end{aligned} \quad (16)$$

Proof: In each case, the first equality is a direct application of property 4 to equation 13, and the second equality is an application of the chain rule to the definition of o^l , taking property 2 into account for the first case. ■

Similarly for the differentials with respect to the inputs of the neural network:

Property 5 Let $\mathcal{G} = (\mathcal{N}, \mathcal{E}, <)$ be a feedforward neural network. For each input node $N^k = In_j$, for each node N^l and for arbitrary input, parameter and local output vectors, x , w and f^k , the following formula holds:

$$\frac{\partial o^{l \rightarrow k}}{\partial x_j}(x, w, f^k) = 0 \quad (17)$$

Intuitively, it implies that the outputs of the nodes depend on an input vector x_j only through the output of the corresponding neuron $N^k = In_j$.

Proof: This result is obvious as x_j never appears in the recursive definition of $o^{l \rightarrow k}$, see definition 8. More formally, we can prove it by recurrence, with the same method as for property 2. ■

And therefore:

Theorem 3 Let $\mathcal{G} = (\mathcal{N}, \mathcal{E}, <)$ be a feedforward neural network. For each input node $N^k = In_j$, for each node N^l and for arbitrary input and parameter vectors x and w , the following formula holds:

$$\frac{\partial o^l}{\partial x_j}(x, w) = \frac{\partial o^{l \rightarrow k}}{\partial o^k}(x, w, o^k(x, w)) \frac{\partial o^k}{\partial x_j}(x, w) \quad (18)$$

$$= \frac{\partial o^{l \rightarrow k}}{\partial o^k}(x, w, o^k(x, w)) dN_i^k(x_j, w^k) \quad (19)$$

Summary:

To compute the differentials of the neural network function, we want to compute the differentials of $o^{l \rightarrow k}$ with respect to its third variable, i.e., $\frac{\partial o^{l \rightarrow k}}{\partial f^k} = \frac{\partial o^{l \rightarrow k}}{\partial o^k}$. The aim of the following section is to show how these differentials can be computed in a back-propagated recursive way.

7.2 Additional notations

To obtain the rank of a node N^k in the predecessor list of its successor N^l , the following definition introduces a new function.

Definition 9 Let $\mathcal{G} = (\mathcal{N}, \mathcal{E}, <)$ be an ordered oriented graph. Let N^k be a node of \mathcal{G} and N^l be a successor of this node. We call $r(k, l)$ the rank of N^k in the predecessor set of N^l .

The next definition generalizes the output of the neural network:

Definition 10 Let $\mathcal{G} = (\mathcal{N}, \mathcal{E}, <)$ be a feedforward neural network. O is by definition the output space of the network and is equal to the product of the output spaces of output nodes. Let S be the product of the output spaces of all nodes of the network, i.e., $S = \prod_{i=1}^n O^i$. We define:

- the output function, $S(x, w)$ from $I \times P$ into S , with:

$$S(x, w) = (o^1(x, w), o^2(x, w), \dots, o^n(x, w)) \quad (20)$$

- the “free” output function, for an arbitrary node N^l , $S^l(x, w, f^l)$, from $I \times P \times O^l$ into S , with:

$$S^l(x, w, f^l) = (o^{1 \rightarrow l}(x, w, f^l), \dots, o^{n \rightarrow l}(x, w, f^l)), \quad (21)$$

which implies $S^l(x, w, o^l(x, w)) = S(x, w)$.

Léon Bottou and Patrick Gallinari used in [3] a lagrangian formalism to prove a restricted version of the extended back-propagation. This method gives a short proof which is unfortunately quite incomplete in the paper and in Léon Bottou’s thesis ([2]). We will use the same method, but we intend to establish a rigorous proof. We need now to introduce a scalar notation:

Let $\mathcal{G} = (\mathcal{N}, \mathcal{E}, <)$ be a feedforward neural network. We call n^l the dimension of the vectorial space O^l (i.e., the output space of neuron N^l). For an arbitrary input vector x and for an arbitrary parameter vector w , $o_j^l(x, w)$ is the j -th coordinate of the vectorial output of node N^l .

$$o^l(x, w) = (o_1^l(x, w), o_2^l(x, w), \dots, o_{n^l}^l(x, w))$$

- Let N^l be an inner node. Then:

$$o^l(x, w) = N^l \left(\left(o_1^{P(l)_1}(x, w), \dots, o_{n^{P(l)_1}}^{P(l)_1}(x, w) \right), \dots, \left(o_1^{P(l)_{p^k}}(x, w), \dots, o_{n^{P(l)_{p^k}}}^{P(l)_{p^k}}(x, w) \right), w^k \right)$$

Let N_j^l be the j -th coordinate function. This function depends on many real inputs. The following slightly incorrect notation will be used:

$$\frac{\partial N_j^l}{\partial o_i^k},$$

for the partial differential of N_j^l with respect to the i -th coordinate of the k -th input vector of N^l .

- Let $N^l = In_i$ be an input neuron. $In(l) = i$ is the rank of N^l as an element of In , so that $In_{In(l)} = N^l$. Let x be an input vector, with $x = (x_1, \dots, x_{in})$. Let $x_{k,l}$ be the l -th coordinate of x_k . Moreover, let in_i be the dimension of I^{In_i} , the input space of node In_i . Then:

$$o^l(x, w) = N^l \left((x_{i,1}, \dots, x_{i,in_i}), w^l \right)$$

The partial differential of N_j^l with respect to the k -th coordinate of it's input vector will be written:

$$\frac{\partial N_j^l}{\partial x_{i,k}}$$

7.3 The main result

Theorem 4

Let $\mathcal{G} = (\mathcal{N}, \mathcal{E}, <)$ be a feedforward neural network. Let x be an arbitrary input vector. Let w be an arbitrary parameter vector and let N^l and N^k be arbitrary nodes. Then:

- If $N^k = N^l$:

$$\frac{\partial o^{l \rightarrow k}}{\partial o^k} (x, w, o^k(x, w)) = Id_{O^k} \quad (22)$$

- If $N^k \notin P^*(N^l)$:

$$\frac{\partial o^{l \rightarrow k}}{\partial o^k} (x, w, o^k(x, w)) = 0_{O^k, O^l}, \quad (23)$$

- If $N^k \in P^+(N^l)$:

$$\begin{aligned} \frac{\partial o^{l \rightarrow k}}{\partial o^k} (x, w, o^k(x, w)) = & \quad (24) \\ & \sum_{N^j \in S(k)} \frac{\partial o^{l \rightarrow j}}{\partial o^j} (x, w, o^j(x, w)) dN_{i_{r(k,j)}}^j \left(o^{P(j)1}(x, w), \dots, o^{P(j)p^j}(x, w), w^j \right) \end{aligned}$$

Proof: The first equation is really obvious, as $o^{k \rightarrow k}(x, w, f^k) = f^k$. The second equation is also obvious, as a consequence of property 3. Let us now study the third case in which N^k is a non output node.

In order to prove this result, we will use a lagrangian method, following Léon Bottou and Patrick Gallinari ([3]). Let τ and λ be two vectors in S (the generalized output space of the neural network). As elements of this space, they are in fact “vector of vectors”. Therefore, we can consider τ^k which belongs to O^k . Let then τ_j^k be the j -th coordinate of τ^k .

Let us now introduce the lagrangian function L_i^l relating to the node N^l . This is a function from $I \times W \times S \times S$ into \mathbb{R} , defined by:

$$\begin{aligned} L_i^l(x, w, \tau, \lambda) = \tau_i^l & - \sum_{k \in P^*(l), k \notin In} \sum_{j=1}^{n^k} \lambda_j^k \left(\tau_j^k - N_j^k \left(\tau^{P(k)1}, \dots, \tau^{P(k)p^k}, w^k \right) \right) \\ & - \sum_{k \in P^*(l) \cap In} \sum_{j=1}^{n^k} \lambda_j^k \left(\tau_j^k - N_j^k (x_{In(k)}, w^k) \right) \end{aligned}$$

There are n^l lagrangian functions for each node N^l , one for each dimension of O^l . The first part of the lagrangian (i.e., τ_i^l) is the studied function and the second part (i.e., $\tau_j^k - N_j^k(\dots)$) corresponds to the constraints for the variables. Obviously, if $\tau^l = o^l(x, w)$ (i.e., if $\tau = S(x, w)$), then the constraints are equal to zero. The reciprocal is also true because the constraints exactly correspond to the propagation formulae. If the constraints are satisfied (equal to zero), then:

$$\forall \lambda, L_i^l(x, w, \tau, \lambda) = o_i^l(x, w)$$

Since the constraints are equivalent to $\tau = S(x, w)$:

$$\forall \lambda, L_i^l(x, w, S(x, w), \lambda) = o_i^l(x, w)$$

Computing the differential of this equation with respect to w_j^k (the j -th coordinate of the k -th parameter vector) leads to:

$$\forall x, w, \lambda, \frac{\partial o_i^l}{\partial w_j^k}(x, w) = \frac{\partial L_i^l}{\partial w_j^k}(x, w, S(x, w), \lambda) + \frac{\partial L_i^l}{\partial \tau}(x, w, S(x, w), \lambda) \frac{\partial S}{\partial w_j^k}(x, w)$$

The first term of the sum is:

- if $k \in P^*(l)$, $k \notin In$

$$\frac{\partial L_i^l}{\partial w_j^k}(x, w, \tau, \lambda) = \sum_{p=1}^{n^k} \lambda_p^k \frac{\partial N_p^k}{\partial w_j^k}(\tau^{P(k)_1}, \dots, \tau^{P(k)_{p^k}}, w^k)$$

- if $k \in P^*(l) \cap In$

$$\frac{\partial L_i^l}{\partial w_j^k}(x, w, \tau, \lambda) = \sum_{p=1}^{n^k} \lambda_p^k \frac{\partial N_p^k}{\partial w_j^k}(x_{In(k)}, w^k)$$

Let us now study $\frac{\partial L_i^l}{\partial \tau}$.

If $k \notin P^*(l)$, then:

$$\frac{\partial L_i^l}{\partial \tau^k}(x, w, \tau, \lambda) = 0,$$

because τ^k does not appear in the definition of L_i^l .

If $k \in P^*(l)$, different cases can occur:

- If $k = l$, then:
 - If $j \neq i$, then:

$$\frac{\partial L_i^l}{\partial \tau_j^l}(x, w, \tau, \lambda) = -\lambda_j^l,$$

because τ_j^l does not appear as input of a neuron (i.e., in the right part of the constraints), since if there was some $k \in P^*(l)$ such that $P(k)_q = N^l$, then the graph would be cyclic.

- If $j = i$, then:

$$\frac{\partial L_i^l}{\partial \tau_i^l}(x, w, \tau, \lambda) = 1 - \lambda_j^l,$$

- If $k \neq l$, then:

$$\frac{\partial L_i^l}{\partial \tau_j^k}(x, w, \tau, \lambda) = -\lambda_j^k + \sum_{q \in S(k) \cap P^*(l)} \sum_{p=1}^{n^q} \lambda_p^q \frac{\partial N_p^q}{\partial \tau_j^{(k,q)}}(\tau^{P(q)_1}, \dots, \tau^{P(q)_{p^q}}, w^q)$$

Let us now define $C_i^l(w, \tau)$, an element of S :

- If $k \notin P^*(l)$, then for all j :

$$C_i^l(w, \tau)_j^k = 0$$

- if $k = l$, then for all j :

$$C_i^l(w, \tau)_j^l = \delta_{ij}$$

- if $k \in P^+(l)$, then for all j :

$$C_i^l(w, \tau)_j^k = \sum_{q \in S(k) \cap P^*(l)} \sum_{p=1}^{n^q} C_i^l(w, \tau)_p^q \frac{\partial N_p^q}{\partial \sigma_j^{\tau(k,q)}} \left(\tau^{P(q)_1}, \dots, \tau^{P(q)_{p^q}}, w^q \right)$$

It is clear that this definition allows to build an element of S , provided that it is applied backward from the last layer to the first layer of the neural network.

The definition of $C_i^l(w, \tau)$ implies that:

$$\frac{\partial L_i^l}{\partial \tau} (x, w, \tau, C_i^l(w, \tau)) = 0$$

Therefore:

$$\frac{\partial o_i^l}{\partial w_j^k} (x, w) = \frac{\partial L_i^l}{\partial w_j^k} (x, w, S(x, w), C_i^l(w, S(x, w))),$$

that is:

- if $k \notin In$:

$$\frac{\partial o_i^l}{\partial w_j^k} (x, w) = \sum_{p=1}^{n^k} C_i^l(w, S(x, w))_p^k \frac{\partial N_p^k}{\partial w_j^k} \left(o^{P(k)_1}(x, w), \dots, o^{P(k)_{p^k}}(x, w), w^k \right)$$

- if $k \in In$

$$\frac{\partial o_i^l}{\partial w_j^k} (x, w) = \sum_{p=1}^{n^k} C_i^l(w, S(x, w))_p^k \frac{\partial N_p^k}{\partial w_j^k} (x_{In(k)}, w^k)$$

These formulae are very close to equations 15 and 16 of theorem 2. Indeed equation 15 (for instance) can be written with scalar notations:

$$\frac{\partial o_i^l}{\partial w_j^k} (x, w) = \sum_{p=1}^{n^k} \frac{\partial o_i^{l \rightarrow k}}{\partial o_p^k} (x, w, o^k(x, w)) \frac{\partial N_p^k}{\partial w_j^k} \left(o^{P(k)_1}(x, w), \dots, o^{P(k)_{p^k}}(x, w), w^k \right)$$

The case $k \in In$ leads to a similar comparison.

Then, the following lemma is introduced:

Lemma 4.1

$$\forall i, l, p, k, \quad C_i^l(w, S(x, w))_p^k = \frac{\partial o_i^{l \rightarrow k}}{\partial o_p^k} (x, w, o^k(x, w))$$

The proof of this lemma will be given at the end of the main proof.

If $k \in P^+(l)$, the definition of $C_i^l(w, S(x, w))_j^k$ and lemma 4.1 imply that:

$$\frac{\partial o_i^{l \rightarrow k}}{\partial o_j^k}(x, w, o^k(x, w)) = \sum_{q \in S(k) \cap P^*(l)} \sum_{p=1}^{n^q} \frac{\partial o_i^{l \rightarrow q}}{\partial o_p^q}(x, w, o^q(x, w)) \frac{\partial N_p^q}{\partial o_j^{r(k, q)}}(o^{P(q)1}(x, w), \dots, o^{P(q)p^q}(x, w), w^q)$$

In order to return to vectorial notations, let us recall that:

$$\frac{\partial o_i^{l \rightarrow k}}{\partial o^k} = \left(\frac{\partial o_i^{l \rightarrow k}}{\partial o_1^k}, \dots, \frac{\partial o_i^{l \rightarrow k}}{\partial o_{n^k}^k} \right),$$

and

$$\frac{\partial N^q}{\partial o_j^r} = \begin{pmatrix} \frac{\partial N_1^q}{\partial o_j^r} \\ \frac{\partial N_2^q}{\partial o_j^r} \\ \dots \\ \frac{\partial N_{n^q}^q}{\partial o_j^r} \end{pmatrix},$$

Therefore:

$$\sum_{p=1}^{n^q} \frac{\partial o_i^{l \rightarrow q}}{\partial o_p^q}(x, w, o^q(x, w)) \frac{\partial N_p^q}{\partial o_j^{r(k, q)}}(o^{P(q)1}(x, w), \dots, o^{P(q)p^q}(x, w), w^q),$$

is exactly the scalar product:

$$\frac{\partial o_i^{l \rightarrow q}}{\partial o^q}(x, w, o^q(x, w)) \frac{\partial N^q}{\partial o_j^{r(k, q)}}(o^{P(q)1}(x, w), \dots, o^{P(q)p^q}(x, w), w^q),$$

which is the term (i, j) of the matrix:

$$\frac{\partial o^{l \rightarrow q}}{\partial o^q}(x, w, o^q(x, w)) \frac{\partial N^q}{\partial o^{r(k, q)}}(o^{P(q)1}(x, w), \dots, o^{P(q)p^q}(x, w), w^q)$$

It leads to:

$$\frac{\partial o^{l \rightarrow k}}{\partial o^k}(x, w, o^k(x, w)) = \sum_{q \in S(k) \cap P^*(l)} \frac{\partial o^{l \rightarrow q}}{\partial o^q}(x, w, o^q(x, w)) \frac{\partial N^q}{\partial o^{r(k, q)}}(o^{P(q)1}(x, w), \dots, o^{P(q)p^q}(x, w), w^q)$$

If $q \notin P^*(l)$:

$$\frac{\partial o^{l \rightarrow q}}{\partial o^q}(x, w, \tau^q) = 0,$$

therefore:

$$\frac{\partial o^{l \rightarrow k}}{\partial o^k}(x, w, o^k(x, w)) = \sum_{q \in S(k)} \frac{\partial o^{l \rightarrow q}}{\partial o^q}(x, w, o^q(x, w)) \frac{\partial N^q}{\partial o^{r(k, q)}}(o^{P(q)1}(x, w), \dots, o^{P(q)p^q}(x, w), w^q),$$

which is exactly equation 24. ■

Proof of lemma 4.1: A new lagrangian function is introduced (for $N^k \in P^+(l)$):

$$\begin{aligned} M_i^{l \rightarrow k}(x, w, \tau, \lambda) = & \tau_i^l - \sum_{q \in P^*(l), q \notin I_n, q \neq k} \sum_{j=1}^{n^q} \lambda_j^q \left(\tau_j^q - N_j^q \left(\tau^{P(q)1}, \dots, \tau^{P(q)p^q}, w^q \right) \right) \\ & - \sum_{q \in P^*(l) \cap I_n, q \neq k} \sum_{j=1}^{n^q} \lambda_j^q \left(\tau_j^q - N_j^q(x_{I_n(q)}, w^q) \right) \end{aligned}$$

Its constraints are fulfilled if and only if:

$$\forall q, \tau^q = o^{q \rightarrow k}(x, w, \tau^k),$$

and therefore:

$$M_i^{l \rightarrow k}(x, w, \tau, \lambda) = o_i^{l \rightarrow k}(x, w, \tau^k)$$

For arbitrary values of τ^k , x , w and λ , it implies that:

$$M_i^{l \rightarrow k}(x, w, S^k(x, w, \tau^k), \lambda) = o_i^{l \rightarrow k}(x, w, \tau^k)$$

Therefore, for any x , w , τ^k and λ :

$$\frac{\partial o_i^{l \rightarrow k}}{\partial o^k}(x, w, \tau^k) = \frac{\partial M_i^{l \rightarrow k}}{\partial \tau}(x, w, S^k(x, w, \tau^k), \lambda) \frac{\partial S^k}{\partial \tau^k}(x, w, \tau^k)$$

Furthermore, for $q \neq k$ and $q \notin P(k)$:

$$\frac{\partial M_i^{l \rightarrow k}}{\partial \tau^q}(x, w, \tau, \lambda) = \frac{\partial L_i^l}{\partial \tau^q}(x, w, \tau, \lambda)$$

Therefore:

$$\frac{\partial M_i^{l \rightarrow k}}{\partial \tau^q}(x, w, \tau, C_i^l(w, \tau)) = 0$$

On the other hand:

$$\frac{\partial M_i^{l \rightarrow k}}{\partial \tau_j^k}(x, w, \tau, \lambda) = \sum_{q \in S(k) \cap P^*(l)} \sum_{p=1}^{n^q} \lambda_p^q \frac{\partial N_p^q}{\partial o_j^{r(k,q)}}(\tau^{P(q)_1}, \dots, \tau^{P(q)_{p^q}}, w^q)$$

The definition of $C_i^l(w, \tau)$ means therefore that:

$$\frac{\partial M_i^{l \rightarrow k}}{\partial \tau_j^k}(x, w, \tau, C_i^l(w, \tau)) = C_i^l(w, \tau)_j^k$$

Finally, we have:

$$\frac{\partial S^k}{\partial \tau^k}(x, w, \tau^k) = \left(\frac{\partial o^{1 \rightarrow k}}{\partial o^k}(x, w, \tau^k), \frac{\partial o^{2 \rightarrow k}}{\partial o^k}(x, w, \tau^k), \dots, \frac{\partial o^{n \rightarrow k}}{\partial o^k}(x, w, \tau^k) \right)$$

Therefore:

$$\frac{\partial o_i^{l \rightarrow k}}{\partial o^k}(x, w, \tau^k) = \sum_{q=1}^n \frac{\partial M_i^{l \rightarrow k}}{\partial \tau^q}(x, w, S^k(x, w, \tau^k), \lambda) \frac{\partial o^{q \rightarrow k}}{\partial o^k}(x, w, \tau^k)$$

As the network is not cyclic, $q \in P(k)$ implies $k \notin P^*(q)$, and therefore $\frac{\partial o^{q \rightarrow k}}{\partial o^k}(x, w, \tau^k) = 0$.

Moreover, if $q \neq k$ and $q \notin P(k)$, $\frac{\partial M_i^{l \rightarrow k}}{\partial \tau^q}(x, w, \tau, C_i^l(w, \tau)) = 0$, therefore:

$$\frac{\partial o_i^{l \rightarrow k}}{\partial o^k}(x, w, \tau^k) = \frac{\partial M_i^{l \rightarrow k}}{\partial \tau^k}(x, w, S^k(x, w, \tau^k), C_i^l(w, S^k(x, w, \tau^k))) \frac{\partial o^{k \rightarrow k}}{\partial o^k}(x, w, \tau^k),$$

i.e.,

$$\frac{\partial o_i^{l \rightarrow k}}{\partial o^k}(x, w, \tau^k) = (C_i^l(w, S^k(x, w, \tau^k))_1^k, C_i^l(w, S^k(x, w, \tau^k))_2^k, \dots, C_i^l(w, S^k(x, w, \tau^k))_{n^k}^k)$$

Therefore, we have:

$$C_i^l(w, S^k(x, w, \tau^k))_j^k = \frac{\partial o_i^{l \rightarrow k}}{\partial o_j^k}(x, w, \tau^k),$$

which implies:

$$C_i^l(w, S(x, w))_j^k = \frac{\partial o_i^{l \rightarrow k}}{\partial o_j^k}(x, w, o^k(x, w))$$

■

7.4 Output functions

Computing the differential of a neural network function is often useful to train this network to perform a given task. A supervised learning method aims at decreasing an error, which is indeed a distance between a desired output and the actual output of the neural network. The error computation is modelled by an output function F which maps the vectorial output of the neural network to another output (the most common case is a real output). The aim of this section is to prove how the extended back-propagation algorithm can be changed to handle such an output function (let us note that the direct method is not modified by an output function).

Theorem 5 *Let $\mathcal{G} = (\mathcal{N}, \mathcal{E}, <)$ be a differentiable feedforward neural network. Let F be a differentiable vectorial function of vectorial variables and let S^1, \dots, S^{out} be its input spaces, with $S^k \simeq O^{Out_k}$. A composite function $F_G(x, w)$ can be defined as:*

$$F_G(x, w) = F\left(o^{Out_1}(x, w), \dots, o^{Out_{out}}(x, w)\right) \quad (25)$$

F is an output function for the network. Let $F_G^k(x, w, f^k)$ be defined in a similar way as G^k (i.e., with a “free” output for node N^k).

If dF_{i_k} stands for the partial differential of F with respect to its k -th variable, then:

$$\frac{\partial F_G}{\partial w^k}(x, w) = \frac{\partial F_G^k}{\partial o^k}(x, w, o^k(x, w)) dN_w^k\left(o^{P(k)_1}(x, w), \dots, o^{P(k)_{p^k}}(x, w), w^k\right) \quad (26)$$

$$\frac{\partial F_G}{\partial x_j}(x, w) = \frac{\partial F_G^k}{\partial o^k}(x, w, o^k(x, w)) dN_i^k(x_j, w^k), \text{ for } N^k = In_j \text{ an input node,} \quad (27)$$

with,

- for $N^k = Out_j$, an output node:

$$\frac{\partial F_G^k}{\partial o^k}(x, w, o^k(x, w)) = dF_{i_j}\left(o^{Out_1}(x, w), \dots, o^{Out_{out}}(x, w)\right), \quad (28)$$

- for $N^k \notin Out$:

$$\begin{aligned} \frac{\partial F_G^k}{\partial o^k}(x, w, o^k(x, w)) = & \\ & \sum_{N^j \in S(k)} \frac{\partial F_G^j}{\partial o^j}(x, w, o^j(x, w)) dN_{i_r(k,j)}^j\left(o^{P(j)_1}(x, w), \dots, o^{P(j)_{p^j}}(x, w), w^j\right) \end{aligned} \quad (29)$$

Proof: F may be considered as a neuron without any parameter. Let N^{n+1} be this neuron. It allows to define an extended neural network $\mathcal{H} = (\mathcal{Q}, \mathcal{F}, <_{\mathcal{H}})$, where $\mathcal{Q} = \mathcal{N} \cup \{\mathcal{N}^{\wedge +\infty}\}$ and $\mathcal{F} = \mathcal{E} \cup \{(\mathcal{O} \sqcup \sqcup_{\parallel}, \mathcal{N}^{\wedge +\infty}) \mid \mathcal{O} \sqcup \sqcup_{\parallel} \in \mathcal{O} \sqcup \sqcup\}$. As no connection is added among the old nodes, $\forall k \leq n, <_{\mathcal{H}}(N^k) = <(N^k)$. For the new node, $P(N^{n+1})_k = Out_k$. The input space of N^{n+1} is $O^{Out_1} \times \dots \times O^{Out_{out}}$, its parameter space is $\{0\}$, and its output space is O^F . It computes the function F . Therefore, we have $H(x, w) = F_G(x, w)$.

We add a H subscript to values computed by \mathcal{H} , and a G subscript to values computed by \mathcal{G} . It is obvious that:

$$\forall l \leq n, o_H^l(x, w) = o_G^l(x, w),$$

and:

$$\forall l \leq n \text{ and } \forall k \leq n, o_H^{l \rightarrow k}(x, w) = o_G^{l \rightarrow k}(x, w)$$

Therefore:

$$\forall k \leq n, F_G^k(x, w, f^k) = o_H^{n+1 \rightarrow k}(x, w, f^k)$$

Theorem 2 can be applied to \mathcal{H} . For $N^k \neq N^{n+1}$:

$$\frac{\partial o_H^{n+1}}{\partial w^k}(x, w) = \frac{\partial o_H^{n+1 \rightarrow k}}{\partial w^k}(x, w, o_H^k(x, w)) dN_w^k \left(o_H^{P_H(k)1}(x, w), \dots, o_H^{P_H(l)pk}(x, w), w^k \right)$$

This formula is exactly equation 26 because $o_H^{n+1}(x, w) = F_G(x, w)$.

Theorem 3 implies for $N^k = In_j$:

$$\frac{\partial o_H^{n+1}}{\partial x_j}(x, w) = \frac{\partial o_H^{n+1 \rightarrow k}}{\partial o^k}(x, w, o_H^k(x, w)) dN_i^k(x_j, w^k)$$

This formula is exactly equation 27.

Finally, the back-propagation theorem may be applied to \mathcal{H} .

Since $P_H^+(N^{n+1}) = \mathcal{N}$, the following formula holds for $N^k \neq N^{n+1}$:

$$\begin{aligned} \frac{\partial o_H^{n+1 \rightarrow k}}{\partial o^k}(x, w, o_H^k(x, w)) = \\ \sum_{N^j \in S_H(k)} \frac{\partial o_H^{n+1 \rightarrow j}}{\partial o^j}(x, w, o_H^j(x, w)) dN_{r(k,j)}^j \left(o_H^{P_H(j)1}(x, w), \dots, o_H^{P_H(j)pj}(x, w), w^j \right) \end{aligned}$$

If N^k is not an output node of \mathcal{G} , then $S_H(k) = S_G(k)$. Therefore, the previous equation implies equation 29.

If $N^k = Out_j$, an output node of \mathcal{G} , then $S_H(k) = N^{n+1}$. Therefore:

$$\begin{aligned} \frac{\partial o_H^{n+1 \rightarrow k}}{\partial o^k}(x, w, o_H^k(x, w)) = \\ \frac{\partial o_H^{n+1 \rightarrow n+1}}{\partial o^{n+1}}(x, w, o_H^j(x, w)) dN_j^{n+1} \left(o_H^{P_H(n+1)1}(x, w), \dots, o_H^{P_H(n+1)pn+1}(x, w), w^{n+1} \right) \end{aligned}$$

The back propagation theorem applies to N^{n+1} , hence:

$$\frac{\partial o_H^{n+1 \rightarrow n+1}}{\partial o^{n+1}}(x, w, o_H^{n+1}(x, w)) = Id_{O^{n+1}}$$

Moreover $P_H(n+1)_l = Out_l$, and therefore:

$$\frac{\partial o_H^{n+1 \rightarrow k}}{\partial o^k}(x, w, o_H^k(x, w)) = dN_j^{n+1} \left(o_H^{Out_1}(x, w), \dots, o_H^{Out_{out}}(x, w) \right),$$

that is:

$$\frac{\partial F_G^k}{\partial o^k}(x, w, o_G^k(x, w)) = dF_{i_j} \left(o_G^{Out_1}(x, w), \dots, o_G^{Out_{out}}(x, w) \right),$$

which is exactly equation 28. ■

Intuitively, this theorem means that adding a function at the end of the neural network only modifies the starting values of the back-propagation process.

8 Input Sharing

In our model, there is a great difference between the input nodes and the non-input nodes. Indeed, two non-input nodes can share a common input, whereas each input node has got its own input vector. In the MLP model, each neuron of the first layer receives the same input. Therefore, our model is not yet able to implement the standard MLP.

In order to avoid this problem, a *sharing function* allows several input neurons to share a common input vector.

Definition 11 Let $\mathcal{G} = (\mathcal{N}, \mathcal{E}, <)$ be a feedforward neural network. Let NI be a vectorial space on \mathbb{R} of finite dimension and let SF be a function from NI into I , the input space of the neural network. SF is called a **sharing function** for the neural network \mathcal{G} . It defines a new function G_{SF} from $NI \times W$ into O as:

$$G_{SF}(y, w) = G(SF(y), w) \quad (30)$$

The following example shows how implement the input sharing of the standard MLP model.

Example 3 Let $\mathcal{G} = (\mathcal{N}, \mathcal{E}, <)$ be a feedforward neural network and let \mathbb{R}^k be the input space of every input neuron. The sharing function SF from \mathbb{R}^k into I is defined as follows:

$$SF(x) = \underbrace{(x, x, \dots, x)}_{\text{in times}}$$

G_{SF} allows each input node to receive its own copy of the input vector x .

Provided that SF is differentiable, the back-propagation method can be used, since G_{SF} can be computed with a slightly modified neural network (SF can be represented as a neuron with no parameter, which output is distributed among the input neurons of \mathcal{G} thanks to projection neurons). It can be proved that the back-propagation computes exactly the same differentials as the simple chain rule for this extended neural network, but in a very unefficient way.

The most efficient computation of the differential with respect to the input uses the chain rule:

$$dG_{SF_i}(x, w) = dG_i(SF(y), w)dSF(y)$$

where dG_i may be computed either with the help of the direct method, or with the help of the back-propagation.

For the differential with respect to the network parameters, $dG_{SF_w}(x, w) = dG_w(SF(y), w)$, i.e., the partial differential of G_{SF} with respect to its parameter vector w is equal to the partial differential of G . Therefore, input sharing does not change anything to standard application of neural networks, since the main application of differential computation for neural networks is parameter learning, which only needs the gradient of an error function with respect to the parameter vector of the network.

9 Complexity

The aim of this section is to compare the theoretical time needed by both differentiation algorithms.

9.1 Notations and preliminary remarks

Both algorithms need to know the input, the output and the differentials of each node. Therefore, the comparison will focus on the cost of the algebraic operations required by both methods.

We introduce the following quantities:

- the main computation load is mostly due to the numerical operations needed for the algebraic operations, i.e., floating point number additions and multiplications. We will assume that the time needed to perform such an operation is 1 (experiments on a Sparc Station 2 show that the time to perform these operations is approximately the same for both addition and multiplication). This is therefore the unit of our formulae.
- $m(i, j, k)$ is the time needed to multiply a (i, j) -matrix and a (j, k) -matrix (approximately $ik(2j - 1)$);
- $s(i, j)$ is the time needed to sum to (i, j) -matrices (approximately ij).

Let $|\cdot|$ be the function which maps a vectorial space to its dimension (for instance, $|O^l|$ is the dimension of the output space of node N^l). The same notation will be used for the number of elements of a finite set (for instance $|\mathcal{N}| = \setminus$).

9.2 Direct algorithm

Theorem 6 *Let $\mathcal{G} = (\mathcal{N}, \mathcal{E}, <)$ be a differentiable feedforward neural network. With the direct algorithm, computing the differential of G with respect to its parameter vector needs a time equal to:*

$$\sum_{N^j \notin \text{In}} \sum_{N^l \in P^+(j)} \left((|P(j) \cap S^*(l)| - 1) s(|O^j|, |W^l|) + \sum_{N^k \in P(j) \cap S^*(l)} m(|O^j|, |O^k|, |W^l|) \right) \quad (31)$$

which is approximately equal to:

$$\sum_{N^j \notin \text{In}} |O^j| \sum_{N^l \in P^+(j)} |W^l| \left(2 \sum_{N^k \in P(j) \cap S^*(l)} |O^k| - 1 \right) \quad (32)$$

Proof: Let N^l and N^h be to distincts nodes. In order to compute $\frac{\partial o^h}{\partial w^l}$, $\frac{\partial o^j}{\partial w^l}$ must be computed for every $N^j \in P^+(N^h) \cap S^*(N^l)$, as it is shown by formula 8.

Computing the differential of G needs $\frac{\partial o^{out_k}}{\partial w^l}$ for all k and l . Since $\mathcal{N} = \bigcup_{\parallel=\infty}^{\parallel} \mathcal{P}^*(\mathcal{O} \parallel \sqcup \parallel)$, $\frac{\partial o^j}{\partial w^l}$ must be computed for all N^j and $N^l \in P^+(N^j)$.

If $N^l \in P^+(N^j)$, $\frac{\partial o^j}{\partial w^l}$ is obtained with the help of formula 8. This computation requires some matrix products and some matrix sums. Each $dN_{i_k}^j$ is a $(|O^j|, |I_k^l|)$ matrix, and each $\frac{\partial o^{P(j)_k}}{\partial w^l}$ is a $(|O^{P(j)_k}|, |W^l|)$ matrix. But $\frac{\partial o^{P(j)_k}}{\partial w^l} = 0$ if $P(j)_k \notin S^*(l)$. Therefore, the cost of the matrix products is:

$$\sum_{N^k \in P(j) \cap S^*(l)} m(|O^j|, |O^k|, |W^l|)$$

The obtained matrices are $(|O^j|, |W^l|)$ matrices and therefore the cost of the sum is:

$$(|P(j) \cap S^*(l)| - 1) s(|O^j|, |W^l|)$$

The total cost is then the sum of these costs for $N^j \notin In$ (if $N^l \in In$, $P(l) = \emptyset$) and for $N^l \in P^+(N^j)$.

■

Corollary 1 Let $\mathcal{G} = (\mathcal{N}, \mathcal{E}, <)$ be a differentiable feedforward neural network. Let F be an output function for \mathcal{G} , with O^F as output space. With the direct algorithm, computing the differential of F_G with respect to its parameter vector requires algebraic operations which total cost is:

$$\sum_{N^j \notin In} |O^j| \sum_{N^l \in P^+(j)} |W^l| \left(2 \sum_{N^k \in P(j) \cap S^*(l)} |O^k| - 1 \right) + |O^F| |W| (2|O| - 1) \quad (33)$$

Proof: The direct method applies the chain rule to the definition of F_G , that is $F \circ G$. The total algebraic cost is the sum of $\frac{\partial G}{\partial w}$ computation cost and of the multiplication cost of $dF(G) \times \frac{\partial G}{\partial w}$. ■

9.3 Back-propagation

Theorem 7 Let $\mathcal{G} = (\mathcal{N}, \mathcal{E}, <)$ be a feedforward neural network. With the back-propagation algorithm, computing the differential of G with respect to its parameter vector needs a time equal to:

$$\sum_{k=1}^{out} \sum_{N^l \in P^+(Out_k)} \left(m(|O^{Out_k}|, |O^l|, |W^l|) + (|S(l) \cap P^*(Out_k)| - 1) s(|O^{Out_k}|, |O^l|) \right. \\ \left. + \sum_{N^j \in S(l) \cap P^+(Out_k)} m(|O^{Out_k}|, |O^j|, |O^l|) \right), \quad (34)$$

approximately equal to:

$$\sum_{k=1}^{out} |O^{Out_k}| \left(|I^{Out_k}| + \sum_{N^l \in P^+(Out_k)} \left(|W^l| (2|O^l| - 1) + |O^l| \left(2 \sum_{N^j \in S(l) \cap P^+(Out_k)} |O^j| - 1 \right) \right) \right) \quad (35)$$

Proof: The back-propagation algorithm includes the recursive computation of the $\frac{\partial o^{j \rightarrow l}}{\partial o^l}$ and the local computation of $\frac{\partial o^l}{\partial w^l}$ for $w^l \in P^+(j)$. Equation 15 implies a computation time equal to $m(|O^j|, |O^l|, |W^l|)$. Moreover, $\frac{\partial o^j}{\partial w^l}$ is required only if $N^j \in Out$. Therefore the local computation cost is:

$$\sum_{k=1}^{out} \sum_{N^l \in P^+(Out_k)} m(|O^{Out_k}|, |O^l|, |W^l|)$$

The value of $\frac{\partial o^{Out_k \rightarrow l}}{\partial o^l}$ is recursively computed with the help of equation 24. $\frac{\partial o^{Out_k \rightarrow j}}{\partial o^j}$ is first multiplied by $dN_{i_r(l,j)}^j$ where j is a successor of l . This computation cost is $m(|O^{Out_k}|, |O^j|, |O^l|)$. If $j \notin P(l) \cap P^*(Out_k)$, then $\frac{\partial o^{Out_k \rightarrow j}}{\partial o^j} = 0$. On the other hand, if $j = Out_k$, then $\frac{\partial o^{Out_k \rightarrow j}}{\partial o^j}$ is the identity matrix. In both cases, no matrix product is required. Therefore, the total cost of the matrix products is:

$$\sum_{N^j \in S(l) \cap P^+(Out_k)} m(|O^{Out_k}|, |O^j|, |O^l|)$$

Afterwards, the sum of the obtained matrices is computed: is:

$$(|S(l) \cap P^*(Out_k)| - 1) s(|O^{Out_k}|, |O^l|)$$

$|S(l) \cap P^*(Out_k)|$ matrices are summed (not only $S(l) \cap P^+(Out_k)$) because when $\frac{\partial o^{Out_k \rightarrow j}}{\partial o^j} = Id$, the resulting product is $dN_{i_r(l,j)}^j$ and must be included in the sum. ■

In the same way:

Theorem 8 *Let $\mathcal{G} = (\mathcal{N}, \mathcal{E}, <)$ be a differentiable feedforward neural network. Let F be an output function for \mathcal{G} , with O^F as output space. With the back-propagation algorithm, computing the differential of $F_{\mathcal{G}}$ with respect to its parameter vector requires algebraic operations which total cost is:*

$$|O^F| \left(\sum_{N^l} |W^l| (2|O^l| - 1) + \sum_{N^l \notin \text{Out}} |O^l| \left(2 \sum_{N^j \in S(l)} |O^j| - 1 \right) \right) \quad (36)$$

Proof: This result is obtained the same way as the previous one (theorem 7). ■

9.4 Application

The obtained complexity formulae are not directly comparable. Without any output function, two classes of multi-layer perceptrons can be determined, so that the direct method is faster than the back-propagation for the first class, whereas the back-propagation should be chosen for the second one.

With an output function, and for models like the multi-layer perceptron, the back-propagation remains the best algorithm, provided that the output function is an error function (i.e., with a real output). For standard extents of the MLP model (such as RBF networks or wavelet networks), the back-propagation is a well suited algorithm.

10 Some examples

Different examples are studied in this section to show how to use our general model. The standard MLP model is described either with neurons as units or with layers as units within this model.

10.1 Neuron-based Multi-Layer Perceptron

The most simple idea to implement a MLP as a particular case of the general model is to consider the standard MLP neuron as a generalized neuron.

Example 4 *A Multi-Layer Perceptron is a neural network \mathcal{G} which fulfils the following conditions:*

1. \mathcal{G} has exactly n layers. For all $k > 1$ and $N \in L^k$, $P(N) = L^{k-1}$. For all $k < n$ and $N \in L^k$, $S(N) = L^{k+1}$. Moreover, $\text{Out} = L^n$.
2. Let l^k be the number of neurons in layer L^k . Then, **every** neuron of layer L^k (with $k > 1$) has exactly l^{k-1} inputs. All corresponding input spaces are equal to \mathbb{R} (real inputs).
3. The output space of **every** neuron of the network is \mathbb{R} (real output).
4. The input space of every neuron of the first layer (i.e., of the input layer) is \mathbb{R}^{l^0} .
5. For a non-input neuron with p inputs, the parameter space is \mathbb{R}^{p+1} .
6. For every input neuron, the parameter space is \mathbb{R}^{l^0+1} .

7. Let N be a non-input node of layer L^k ($k > 1$). Then, for a particular function T from \mathbb{R} into \mathbb{R} , called the transfer function of the neuron, N computes:

$$N(x_1, x_2, \dots, x_{l^{k-1}}, (w_1, w_2, \dots, w_{l^{k-1}+1})) = T\left(\sum_{i=1}^{l^{k-1}} w_i x_i + w_{l^{k-1}+1}\right)$$

8. Let N be an input node. Then, for a particular function T from \mathbb{R} into \mathbb{R} , called the transfer function of the neuron, N computes:

$$N((x_1, x_2, \dots, x_{l^0}), (w_1, w_2, \dots, w_{l^0+1})) = T\left(\sum_{i=1}^{l^0} w_i x_i + w_{l^0+1}\right)$$

9. The network uses a sharing function SF from \mathbb{R}^{l^0} into $\mathbb{R}^{in \times l^0}$. SF is defined as follows:

$$SF(x) = \underbrace{(x, x, \dots, x)}_{\text{in times}}$$

The complexity theorems of section 9 may be applied to this model to estimate the computation cost of the differential with respect to the parameters.

- Without any output function:

– for the direct method:

$$\sum_{k=2}^n l^k \left((2^{l^{k-1}} - 1) \sum_{p=1}^{k-2} \sum_{N^l \in L^p} |W^l| + \sum_{N^l \in L^{k-1}} |W^l| \right) \quad (37)$$

– for the back-propagation method:

$$l^n \left(\sum_{p=1}^{n-1} \sum_{N^l \in L^p} |W^l| + \sum_{p=1}^{n-2} l^p (2^{l^{p+1}} - 1) \right) \quad (38)$$

If $n = 2$ both formulae are equals. We can now study two particular cases:

1. Let the studied MLP have decreasing layers ($l^k \leq l^{k-1}$). The complexity formulae may be expressed thanks to a recursion. The direct computation time is D_n , where:

$$\forall p \leq n, \quad D_p = \sum_{k=2}^p l^k \left((2^{l^{k-1}} - 1) \sum_{q=1}^{k-2} \sum_{N^l \in L^q} |W^l| + \sum_{N^l \in L^{k-1}} |W^l| \right)$$

The back-propagation computation time is B_n , where:

$$\forall p \leq n, \quad B_p = l^p \left(\sum_{q=1}^{p-1} \sum_{N^l \in L^q} |W^l| + \sum_{q=1}^{p-2} l^q (2^{l^{q+1}} - 1) \right)$$

Then:

$$\begin{aligned}
B_2 &= D_2 \\
D_{p+1} &= D_p + l^{p+1} \left((2l^p - 1) \sum_{q=1}^{p-1} \sum_{N^l \in L^q} |W^l| + \sum_{N^l \in L^p} |W^l| \right) \\
B_{p+1} &= l^{p+1} \left(\frac{B_p}{l^p} + \sum_{N^l \in L^p} |W^l| + l^{p-1} (2l^p - 1) \right)
\end{aligned}$$

For each N^l , $|W^l| \geq 1$. Let us prove that $D_n \geq B_n$, by means of a recursion. First, $D_2 = B_2$. Let us assume that $p \geq 2$ and $k \leq p \Rightarrow D_k \geq B_k$. Since $l^{p+1} \leq l^p$, $l^{p+1} \frac{B_p}{l^p} \leq B_p$. Therefore, $l^{p+1} \frac{B_p}{l^p} \leq D_p$. Moreover, $|W^l| \geq 1$ implies:

$$(2l^p - 1) \sum_{q=1}^{p-1} \sum_{N^l \in L^q} |W^l| \geq (2l^p - 1) \sum_{q=1}^{p-1} l^q \geq (2l^p - 1) l^{p-1}$$

Therefore:

$$D_{p+1} \geq D_p + l^{p+1} (2l^p - 1) l^{p-1} + l^{p+1} \sum_{N^l \in L^p} |W^l|$$

Hence $D_{p+1} \geq B_{p+1}$.

For a MLP with decreasing layers, the back-propagation algorithm is faster than the direct method.

2. Let us now study the case of a bottleneck: in the studied MLP, there is a layer with only one neuron, and this layer precedes a layer with many neurons.

The complexity formulae may be written as follows:

$$\begin{aligned}
D_n &= \alpha + l^n \beta \\
B_n &= l^n \gamma \\
\alpha &= D_{n-1} \\
\beta &= \sum_{N^l \in L^{n-1}} |W^l| + (2l^{n-1} - 1) \sum_{p=1}^{n-2} \sum_{N^l \in L^p} |W^l| \\
\gamma &= \sum_{p=1}^{n-1} \sum_{N^l \in L^p} |W^l| + \sum_{p=1}^{n-2} l^p (2l^{p+1} - 1)
\end{aligned}$$

In a MLP, the number of neurons in layer n does not influence the number of parameters of the neurons that belong to the preceding layers. Therefore, α , β and γ do not depend on l^n . If $l^{n-1} = 1$, then:

$$\beta - \gamma = - \sum_{p=1}^{n-2} l^p (2l^{p+1} - 1)$$

Therefore, if $n > 2$ (and $l^{n-1} = 1$), then there exists $L \in \mathbb{N}$, such that $l^n \geq L \Rightarrow B_n > D_n$.

This type of neural network architecture is really uncommon but it proves that **the back-propagation is not always the best algorithm.**

- With an error function (i.e., a real output function):

– for the direct method:

$$D_e = \sum_{k=2}^n l^k \left((2l^{k-1} - 1) \sum_{p=1}^{k-2} \sum_{N^l \in L^p} |W^l| + \sum_{N^l \in L^{k-1}} |W^l| \right) + (2l^n - 1) \sum_{N^l} |W^l| \quad (39)$$

– for the back-propagation method:

$$B_e = \sum_{p=1}^n \sum_{N^l \in L^p} |W^l| + \sum_{p=1}^{n-1} l^p (2l^{p+1} - 1) \quad (40)$$

Let us recall that $|W^l| \geq 1$ for all N^l and that $l^k \geq 1$ for all k .

1. If $n = 2$:

$$\begin{aligned} D_e &= l^2 \sum_{N^l \in L^1} |W^l| + (2l^2 - 1) \sum_{N^l} |W^l| \\ &= l^2 \sum_{N^l \in L^1} |W^l| + (2l^2 - 1) \left(\sum_{N^l \in L^1} |W^l| + \sum_{N^l \in L^2} |W^l| \right) \\ B_e &= \sum_{N^l} |W^l| + l^1 (2l^2 - 1) \end{aligned}$$

$|W^l| \geq 1 \Rightarrow \sum_{N^l \in L^1} |W^l| \geq l^1$. Therefore $D_e \geq B_e$, since $l^2 \geq 1$.

2. In most neural networks derived from the MLP model such as the RBF network, we can assume that $|W^l| \geq l^{k-1}$ if $N^l \in L^k$ (the previous description shows that it is always true for a MLP). Therefore:

$$\sum_{N^l \in L^p} |W^l| \geq l^p l^{p-1}$$

If $n = 3$, then:

$$\begin{aligned} D_e &\geq l^2 l^1 l^0 + l^3 (2l^2 - 1) l^1 l^0 + l^3 \sum_{N^l \in L^2} |W^l| + (2l^3 - 1) \sum_{N^l} |W^l| \\ B_e &= \sum_{N^l} |W^l| + l^1 (2l^2 - 1) + l^2 (2l^3 - 1) \end{aligned}$$

Moreover:

$$\begin{aligned} l^3 (2l^2 - 1) l^1 l^0 &\geq l^1 (2l^2 - 1) \\ (2l^3 - 1) \sum_{N^l \in L^2} |W^l| &\geq l^2 (2l^3 - 1) \\ (2l^3 - 1) \sum_{N^l \notin L^2} |W^l| + l^3 \sum_{N^l \in L^2} |W^l| &\geq \sum_{N^l} |W^l| \end{aligned}$$

Therefore $D_e \geq B_e$.

3. If $n > 3$:

$$k > 2 \implies \sum_{p=1}^{k-2} \sum_{N^l \in L^p} |W^l| \geq \sum_{p=1}^{k-2} l^p$$

Therefore:

$$D_e \geq 2 \sum_{k=3}^{n-1} l^k l^{k+1} + l^3 (2l^2 - 1) l^1 l^0 + \sum_{k=2}^n l^k l^{k-1} l^{k-2} + (2l^n - 1) \sum_{N^l} |W^l|$$

which implies that:

$$D_e - B_e \geq l^3 (2l^2 - 1) l^1 l^0 + \sum_{k=2}^n l^k l^{k-1} l^{k-2} - l^1 (2l^2 - 1) - l^2 (2l^3 - 1)$$

As $n \geq 4$, we have:

$$\sum_{k=2}^n l^k l^{k-1} l^{k-2} \geq l^1 l^2 l^3 + l^2 l^3 l^4 \geq 2l^2 l^3$$

Moreover, $l^3 (2l^2 - 1) l^1 l^0 \geq l^1 (2l^2 - 1)$. It allows to conclude that $D_e \geq B_e$.

With an error function and with a standard multilayer architecture (MLP, RBF network, wavelet network, ...), the back-propagation algorithm is faster than the direct method.

- For a standard MLP-like neural network, for which $|W^l| = l^{k-1} + 1$ if $N^l \in L^k$, the back-propagation complexity is equal to:

$$\sum_{p=1}^n l^p (l^{p-1} + 1) + \sum_{p=1}^{n-1} l^p (2l^{p+1} - 1) \leq 3 \sum_{p=1}^n l^p (l^{p-1} + 1) \quad (41)$$

Now, the number of parameters of such a neural network is $\sum_{p=1}^n l^p (l^{p-1} + 1)$. A standard result appears: the time needed to compute the gradient with the back-propagation algorithm is proportional to the number of parameters, provided that every neural computation satisfies a similar condition (for any neuron N^k , the local computation of σ^k , dN_w^k and dN_i^k needs a time proportional to $|W^k|$). This condition is satisfied not only by a standard sigmoid neuron but also by a RBF neuron or a wavelet neuron.

10.2 Layer-based Multi-Layer Perceptron

In a MLP, a layer performs a local vectorial transformation. Therefore, such a layer can be considered as a generalized neuron. This point of view leads to the following model:

Example 5 *A n -layer Perceptron is a neural network \mathcal{G} which fulfils the following conditions:*

1. \mathcal{G} has exactly n neurons and its graph is linear, i.e., $\mathcal{E} = \{(\mathcal{N}^{\parallel}, \mathcal{N}^{\parallel+\infty}), \infty \leq \parallel \leq -\infty\}$;
2. neuron N^k has exactly one input space of dimension l^{k-1} ;
3. the output space of neuron N^k is \mathbb{R}^{l^k} ;
4. the parameter space of neuron N^k is $\mathbb{R}^{l^k(l^{k-1}+1)}$;

5. N^k computes the following function:

$$N^k \left((x_1, \dots, x_{l^{k-1}}), \left(p_{(1,1)}, \dots, p_{(1,l^{k-1}+1)}, \dots, p_{(l^k,1)}, \dots, p_{(l^k,l^{k-1}+1)} \right) \right) = \left(T_1^k \left(\sum_{i=1}^{l^{k-1}} p_{(1,i)} x_i + p_{(1,l^{k-1}+1)} \right), \dots, T_{l^k}^k \left(\sum_{i=1}^{l^{k-1}} p_{(l^k,i)} x_i + p_{(l^k,l^{k-1}+1)} \right) \right),$$

where each T_i^k is a transfer function from \mathbb{R} into \mathbb{R} .

Applying the complexity theorems of section 9 leads to the following results for this new MLP model.

- without any error function:

– for the direct algorithm:

$$\sum_{j=2}^n l^j (2l^{j-1} - 1) \sum_{k=1}^{j-1} l^k (l^{k-1} + 1) \quad (42)$$

– for the back-propagation algorithm:

$$l^n \left(\sum_{k=1}^{n-1} l^k (l^{k-1} + 1) (2l^k - 1) + \sum_{k=1}^{n-2} l^k (2l^{k+1} - 1) \right) \quad (43)$$

- with an error function:

– for the direct algorithm:

$$\sum_{j=2}^n l^j (2l^{j-1} - 1) \sum_{k=1}^{j-1} l^k (l^{k-1} + 1) + (2l^n - 1) \sum_{j=1}^n l^j (l^{j-1} - 1) \quad (44)$$

– for the back-propagation algorithm:

$$\sum_{k=1}^n l^k (l^{k-1} + 1) (2l^k - 1) + \sum_{k=1}^{n-1} l^k (2l^{k+1} - 1) \quad (45)$$

In all cases, the computation cost is longer for this model than for the neuron-based MLP model. This result can be simply explained as follows. In the complexity formulae, it is always assumed that the time needed to compute the product of matrix A of size (i, j) and matrix B of size (j, k) is proportional to $ik(2j - 1)$. This is true for arbitrary matrices. But if A or B is in particular form (if A is diagonal for instance), the computation time can be reduced.

Now the local differential dN_w^k has a very particular form within the layer-based MLP model. It has only $l^k(l^{k-1} + 1)$ non zero terms (among $(l^k)^2(l^{k-1} + 1)$ ones). It can be shown that a particular multiplication may be defined for such particular jacobian matrices, so that the implied computation cost is the same for both MLP models.

Once again, RBF layers and wavelet layers define similar jacobian matrices. Therefore, the neuron-based model and the layer-based model lead to the same complexity for the differential computation of most employed multilayer neural networks.

11 Conclusion

Following Léon Bottou and Patrick Gallinari, we have introduced a general model for feedforward neural networks. This model is general enough to handle all standard models derived from the Multi-Layer Perceptron. But it is precise enough to define an extended back-propagation to compute the differential of the vectorial function that the neural network computes.

We have computed the theoretical complexity of the direct method (simply based on the chain rule) and of the back-propagation method. It proves that the choice of the faster algorithm depends on the architecture of the network, though the back-propagation is the faster method in most usual cases.

This general model is a fundamental tool since it provides us with a general mathematical description of feedforward neural networks. It allows therefore a general study of their properties. Moreover this model can be implemented in this general form with the help of an object oriented language ([6]). This implementation makes the designing of a new neural network easier: many methods (such as the back-propagation) are already implemented for any neural network that is derived from the general model, provided that the user defines the local computation the new neurons perform.

References

- [1] E. K. Blum and K. L. Leong. Approximation theory and feedforward networks. *Neural Networks*, 4:511–515, 1991.
- [2] L. Bottou. *Une Approche théorique de l'Apprentissage Connexionniste ; Applications à la reconnaissance de la Parole*. PhD thesis, Université d'Orsay, 1991.
- [3] L. Bottou and P. Gallinari. A Framework for the Cooperation of Learning Algorithms. In R.P. Lippmann, J.E. Moody, and D.S. Touretzky, editors, *Neural Information Processing Systems*, volume 3, pages 781–788. Morgan Kaufman, 1991.
- [4] M. R. J. Fombellida, M. J. M. Minsoul, and J. L. O. Destiné. Perceptrons multi-couches et fonctions d'activation non monotones. In *Proc. Neuro-Nî mes*, pages 321–339, Nî mes, 1990.
- [5] K.-I. Funahashi. On the approximate realization of continuous mappings by neural networks. *Neural Networks*, 2:183–192, 1989.
- [6] C. Gégout, B. Girau, and F. Rossi. NSK, an Object-Oriented Simulator Kernel for Arbitrary Feedforward Neural Networks. In *Int. Conf. on Tools with Artificial Intelligence*, pages 93–104, New Orleans (Louisiana), November 1994. IEEE.
- [7] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.
- [8] V. Kurková. Universal approximation using feedforward neural networks with gaussian bar units. In Bernd Neumann, editor, *Proc. Euro. Conf. on Artificial Intelligence*, pages 193–197, Vienna, Aug. 1992. ECCAI, John Wiley & Sons.
- [9] J.A. Leonard, M. A. Kramer, and L. H. Unger. Using radial basis functions to approximate a function and its error bounds. *IEEE Trans. Neural Networks*, 3(4):624–627, Jul. 1992.
- [10] D. B. Parker. Optimal algorithms for adaptive networks: second order back propagation, second order direct propagation, and second order hebbian learning. In *Proc. First Int. Joint Conf. Neural Networks*, volume II, pages 593–600, November 1987.
- [11] T. Poggio and F. Girosi. Networks for approximation and learning. *Proc. IEEE*, 78(9):1481–1497, September 1990.
- [12] D. Röckmann and C. Moraga. Using quadratic perceptrons to reduce interconnection density in multilayer neural networks. In A. Prieto, editor, *Artificial Neural Networks*, chapter 1, pages 86–92. Springer-Verlag, 1991.
- [13] F. Rossi and C. Gégout. Geometrical Initialization, Parametrization and Control of Multilayer Perceptrons : Application to Function Approximation. In *Int. Conf. on Neural Networks*, volume I, pages 546–550, Orlando (Florida), June 1994. IEEE.
- [14] Q. Zhang and A. Benveniste. Wavelet networks. *IEEE Trans. On Neural Networks*, 3(6):889–898, November 1992.