Generic Back-Propagation in Arbitrary FeedForward Neural Networks*

Cédric Gégout[•◇⋆] Bernard Girau[•] Fabrice Rossi[◇⋆†]

◇ École Normale Supérieure de Paris
45 rue d'Ulm
75005 PARIS France

• École Normale Supérieure de Lyon
L.I.P.
46 avenue d'Italie
69007 LYON France

⋆ THOMSON-CSF
SDC/DPR/R4
7 rue des Mathurins
92223 BAGNEUX France

Abstract

In this paper, we describe a general mathematical model for feedforward neural networks. The final form of the network is a vectorial function f of two variables, x (the input of the network) and w (the weight vector). We show that the differential of f can be computed with an extended back-propagation algorithm or with a direct method. By evaluating the time needed to compute the differential with the help of both methods, we show how to chose the best one. We introduce also input sharing and output function which allow us to implement efficiently a multilayer perceptron with our model.

1 Introduction

Léon Bottou and Patrick Gallinari have proposed in [1] a general framework for the cooperation of neural modules. In this paper we extend this model in order to derive a general model for feedforward neural networks. The model presented in section 2 can simulate multilayer perceptrons and related models, with the help of an extension proposed in section 4. We show in section 3 that an extended back-propagation algorithm allows us to compute the differentials of the function computed by the net and therefore to use gradient based neural training, as explained in section 6. We give in sections 5 and 6 complexity results allowing us to choose between direct calculation and retro-propagation in order to compute efficiently the gradient for a supervised learning.

*Published in ICANNGA'95 Proceedings.
Available at
http://apiacoa.org/publications/1995/icannga95.pdf

†Up to date contact informations for Fabrice Rossi are avaible at http://apiacoa.org/

Due to space limitation, the proofs are ommited in this paper. A technical report gives all the necessary details (see [3]).

2 The general model

2.1 The neuron

In our model, a neuron is a vectorial function of several variables. More formally, we have :

Definition 2.1 *Let n be a positive integer and let $I_1, \ldots, I_n$, W and O be $n+2$ vectorial spaces on $\mathbb{R}$ of finite dimensions. A* **n-input neuron** *is a continuously differentiable function from $I_1 \times I_2 \times \ldots \times I_n \times W$ to O.*

If N is such a neuron, we write dN_w its partial differential with respect to its $(n+1)$-th variable and dN_{i_k} its partial differential with respect to its k-th variable.

In this definition, W, I_k and O are respectively the weight space of the neuron, its$k$-th input space and its output space.

2.2 The neural net

In order to define a neural net, we need to introduce ordered graphs :

Definition 2.2 *An* **ordered graph** *is a triple $(\mathcal{N}, \mathcal{E}, <)$ which fulfills the following conditions :*

- *$\mathcal{N}$ is a* **finite totally ordered** *set of nodes. It is in fact a sequence of nodes and we will write its elements as $N^1, N^2, \ldots$;*
- *$\mathcal{E}$ is a subset of $\mathcal{N}^{\in}$. $e = (x, y) \in \mathcal{E}$ is an* **edge** *of the graph and it represents a connection* from *x* to *y ;*

- $<$ *is a function from* $\mathcal{N}$ *to* $\mathcal{N}^{\in}$*. It associate 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 we can refer to* $Pred(N^i)_k$ *for its* k*-th element.*

In order to simplify the rest of the paper we introduce here some notations : $\mathcal{G} = (\mathcal{N}, \mathcal{E}, <)$ is an ordered graph with exactly n nodes ; $N^1, \ldots, N^n$ 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. We assume that node N^k has exactly p^k predecessors and s^k successors, and that we call $P(k)_1, \ldots, P(k)_{p^k}$ the sequence of the predecessors of N^k and $S(k)_1, \ldots, S(k)_{s^k}$ the sequence of the successors of N^k. In general, superscripts correspond to node numbers and subscripts correspond to input or output numbers. Furthermore, we will not make any distinction between a node N^k and its rank k.

We also need to generalize the notion of predecessor : for an arbitrary node N, we define $P^0(N)$ as $\{N\}$ and recursively $P^k(N)$ as $\{M \in \mathcal{N} \mid \exists \mathcal{Q} \in \mathcal{P}^{\|-\infty}(\mathcal{N})$ so that $(\mathcal{M}, \mathcal{Q}) \in \mathcal{E}\}$. We have therefore $P^1(N) = P(N)$. We call also $P^+(N)$ the set $\cup_{k=1}^{\infty} P^k(N)$ and $P^*(N) = P^0(N) \cup P^+(N)$. Similar sets can be defined in order to generalize the notion of successor.

We can now define a neural network :

Definition 2.3 *A* **feedforward neural network** *is an ordered graph* $\mathcal{G} = (\mathcal{N}, \mathcal{E}, <)$ *fulfilling the following conditions :*

1. *The graph has* **no-cycle**.
2. *The elements of* $\mathcal{N}$ *are* j**-input neurons** *(*j *depends of course on the neuron). The output space of* N^k *is* O^k *and its weight space is* W^k*.*
3. *If* $p^k > 0$ *then neuron* N^k *is a* p^k*-input neuron. That means it has exactly one input for each of its predecessors in the graph (but only if it has predecessors !). The input spaces of* N^k *are* $I^k_1, \ldots, I^k_{p^k}$*.*
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 for each input* i *of the neuron* N^k *the following condition holds :* $\dim I^k_i = \dim O^{P(k)_i}$*.*

Let us now introduce some additionnal definition related to the neural network.

In is the **subset of** $\mathcal{N}$ **which elements have no predecessor**, i.e., $In = \{N \in \mathcal{N} \mid \mathcal{P}(\mathcal{N}) = \emptyset\}$. In has in elements. As a subset of $\mathcal{N}$, In is totally ordered and $In_1, \ldots, In_{in}$ is the ordered sequence of its elements. The elements of In are connected to the "outside" by means of their inputs. As we make no disctinction between a node and its rank, we can call for instance O^{In_k} the output space of the node $N^j = In_k$, which is in fact O^j.

Out is the **subset of** $\mathcal{N}$ **which elements have no successor**, i.e., $Out = \{N \in \mathcal{N} \mid \mathcal{S}(\mathcal{N}) = \emptyset\}$. Out has out elements and is totally ordered. $Out_1, \ldots, Out_{out}$ 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, the vectorial space $O = \prod_{k=1}^{out} O^{Out_k}$ is **the output space** of the neural network and the vectorial space $W = \prod_{k=1}^{n} W^k$ is **the weight space** of the neural network.

2.3 Computing the output

We define the output of the network like this :

Definition 2.4 *Let* $\mathcal{G}$ *be a feedforward neural network. Let* $x = (x_1, \ldots, x_{in}) \in I$ *be an input vector and let* $w = (w^1, \ldots, w^n) \in W$ *be a weight vector. For each* l*,* $1 \leq l \leq n$*,* $o^l(x, w)$*, the output of the neuron* N^l*, is computed with the help of the following recursive construction :*

- *If* $N^l \in In$*, we have* $N^l = In_k$*. Then* $o^l(x, w) = N^l(x_k, w^l)$ *;*
- *If* $N^l \notin In$*, the output of* N^l *is obtained by* $o^l(x, w) = N^l(o^{P(l)_1}(x, w), \ldots, o^{P(l)_{p^l}}(x, w), w^l)$*.*

Finally, $G(x, w)$*, the output of the network, is obtained by* $G(x, w) = (o^{Out_1}(x, w), \ldots, o^{Out_{out}}(x, w))$*.*

Of course this definition is correct only because the underliying graph is non cyclic.

In the rest of the paper, we will call $I^l(x, w)$ the generalized input of node N^l, i.e :

$$(o^{P(l)_1}(x, w), \ldots, o^{P(l)_{p^l}}(x, w), w^l)$$

3 Computing the differential

3.1 Direct method

As a composed function, G is continuously differentiable. The first method to compute its differential is to use the classical derivation rule for composed function. This is the *direct method*.

Theorem 3.1 *Let $\mathcal{G}$ be a feedforward neural network. The o^l functions are continuously differentiable and we have :*

- *if $N^l \neq N^k$:*
  - *if $N^l \notin In$:*

$$\frac{\partial o^l}{\partial w^j}(x,w) = \sum_{k=1}^{p^l} dN^l_{i_k}\left(I^l(x,w)\right)\frac{\partial o^{P(l)_k}}{\partial w^j}(x,w) \quad (1)$$

  - *if $N^l \in In$:* $$\frac{\partial o^l}{\partial w^j}(x,w) = 0 \quad (2)$$

- *if $N^l = N^k$:* $$\frac{\partial o^l}{\partial w^l}(x,w) = dN^l_w\left(I^l(x,w)\right) \quad (3)$$

We have a similar property if we consider $\frac{\partial o^l}{\partial x_i}$, where x_i is the input of the i-th input neuron.

3.2 Back-propagation

The key idea of the back-propagation algorithm is to consider $o^k(x,w)$, the output of neuron N^k, as a function of $o^l(x,w)$, the output of another neuron N^l. In fact, we define a new function $o^{k\to l}(x,w,f^l)$ and we have of course $o^{k\to l}(x,w,o^l(x,w)) = o^k(x,w)$. We have the following theorem :

Theorem 3.2 *Let $\mathcal{G}$ be a feedforward neural network. Let N^k, N^l, x and w be respectively two neurons of $\mathcal{G}$, an arbitrary input vector and an arbitrary weight vector. Then we have :*

$$\frac{\partial o^k}{\partial w^l}(x,w) = \frac{\partial o^{k\to l}}{\partial o^l}\left(x,w,o^l(x,w)\right) dN^l_w\left(I^l(x,w)\right) \quad (4)$$

A similar equation is fullfilled for $\frac{\partial o^l}{\partial x_i}$.

The back-propagation algorithm gives a recursive method to compute $\frac{\partial o^{k\to l}}{\partial o^l}$. We first need an additionnal definition :

Definition 3.1 *Let $\mathcal{G}$ be an ordered graph and let N^k and N^l be two nodes of $\mathcal{G}$. $r(k,l)$ is the rank of N^k in the predecessor set of N^l, i.e., N^k is the $r(k,l)$-th predecessor of N^l.*

Theorem 3.3 *Let $\mathcal{G}$ be a feedforward neural network. Let N^k, N^l, x and w be respectively two neurons of $\mathcal{G}$, an arbitrary input vector and an arbitrary weight vector. Then we have :*

- *if $N^l = N^k$ then :*

$$\frac{\partial o^{k\to l}}{\partial o^l}\left(x,w,o^l(x,w)\right) = Id_{O^k}, \quad (5)$$

where Id_{O^k} is the identity function of O^k the output space of N^k.

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

$$\frac{\partial o^{k\to l}}{\partial o^l}\left(x,w,o^l(x,w)\right) = 0_{O^l,O^k}, \quad (6)$$

where $0_{A,B}$ is the null function from A to B, two vectorial spaces.

- *if $N^l \in P^+(N^k)$ then :*

$$\frac{\partial o^{k\to l}}{\partial o^l}\left(x,w,o^l(x,w)\right) = \quad (7)$$

$$\sum_{N^j\in S(l)} \frac{\partial o^{k\to j}}{\partial o^j}\left(x,w,o^j(x,w)\right) dN^j_{i_{r(l,j)}}\left(I^j(x,w)\right)$$

The formula 7 gives a recursive method for computing $\frac{\partial o^{k\to l}}{\partial o^l}$. In order to compute $\frac{\partial o^{k\to l}}{\partial o^l}$, we need $\frac{\partial o^{k\to j}}{\partial o^j}$ for $N^j \in S(l)$. Therefore, $\frac{\partial o^{k\to l}}{\partial o^l}$ is computed from the last layer of the network to the input layer : this is a backward algorithm nad therefore an extended back-propagation.

4 Input sharing

4.1 The model

The obtained model is rather powerfull but cannot imitate simple models such as multilayer perceptron (MLP). The main problem is that input nodes have different inputs whereas in a MLP they share the same input. In order to avoid this problem, we introduce an input sharing function :

Definition 4.1 *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 to 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 $NI \times W$ to O like this :*

$$G_{SF}(y,w) = G(SF(y),w) \quad (8)$$

By using a replicating function which maps one vector x to a n-tuple which elements are all equal to x, we can give the same input vector to all input neurons and therefore simulate a MLP. In fact, with this last definition, we can implement with our model all classic feedforward neural networks such as MLP, Radial Basis Function networks, wavelet networks, etc. The model is therefore a general representation of feedforward neural networks.

4.2 Differentials

The input sharing method does not change the differential of G with respect to its weight vector. In fact, we have $dG_{SF\,w}(y,w) = dG_w(SF(y),w)$. Therefore, we don't have to take care of a sharing function when evaluating the time needed to compute the differential of G with respect to its weight vector.

5 Complexity

In this section, we give the time needed to compute the matricial operations involved in both calculation methods. These times do not include the time needed to compute the local differential (e.g., dN^l_w) for individual nodes. They take only into account the matricial products and sums needed to compute the differential of the output of the network with respect to its weight vector. In the sequel, $|E|$ is the dimension of the vectorial space E.

5.1 Direct method

Theorem 5.1 *Let $\mathcal{G}$ be a neural network. The time needed to compute the matricial operations required for calculating dG_w with the direct method is proportionnal to :*

$$\sum_{N^j \notin In} |O^j| \sum_{N^l \in P^+(j)} |W^l| \left(-1 + \sum_{N^k \in P(j) \cap S^*(l)} 2|O^k| \right) \quad (9)$$

5.2 Back-propagation

Theorem 5.2 *Let $\mathcal{G}$ be a neural network. The time needed to compute the matricial operations required for calculating dG_w with the back-propagation method is proportionnal to :*

$$\sum_{k=1}^{out} |O^{Out_k}| \left[\sum_{N^l \in P^+(Out_k)} \left|W^l\right| \left(2\left|O^l\right| - 1\right) \right. \quad (10)$$
$$\left. + \sum_{N^l \in P^+(Out_k)} \left|O^l\right| \left(-1 + \sum_{N^j \in S(l) \cap P^+(Out_k)} 2\left|O^j\right| \right) \right]$$

5.3 Comparison

The formulae are not directly comparable. In fact, even for a classic MLP architecture, the number of neurons can be chosen so that the direct method is faster than the back-propagation. Therefore, in order to compute efficiently the differential of the output of a particular network with respect to its weight vector, the theoretical costs have to be compared.

6 Output function

6.1 Model

In order to train a neural network to approximate a function, we use an error function which is in fact a distance between the output of the network and the desired output. This error is a function of the weight vector of the network and gradient based training methods need only the gradient of this function. In order to compute this gradient, we can handle the error E as a composed function of a distance d and the neural output $G(x,w)$. We compute dG_w with an arbitrary method and then use the classic chain rule to obtain ∇E as the result of a matricial product. Let us note that we need to introduce the desired output into the formula.

We can also see the couple (d,y), where d is a distance fonction and y a desired output, as a particular neuron with no weight vector and with *out* inputs (one input for each output neuron of $\mathcal{G}$). We can therefore apply the back-propagation to this model.

6.2 Back-propagation

We obtain the following theorem :

Theorem 6.1 *Let $\mathcal{G}$ be a feedforward neural network and let F be a continuously differentiable vectorial function with out variables from $S^1 \times \ldots \times S^{out}$ to O^F, with $S^i \simeq O^{Out_i}$. Let $F_{\mathcal{G}}(x,w)$ be :*

$$F\left(o^{Out_1}(x,w), \ldots, o^{Out_{out}}(x,w)\right) = F(G(x,w))$$

We define in a similar way $F^l_{\mathcal{G}}$, where N^l is an arbitrary node, with the help of $o^{Out_j \to l}$.

Let dF_{i_k} be the partial differential of F with respect to its k-th variable. We have :

$$\frac{\partial F_{\mathcal{G}}}{\partial w^l}(x,w) = \frac{\partial F^l_{\mathcal{G}}}{\partial o^l}(x,w,o^l(x,w)) dN^l_w(I^l(x,w)) \quad (11)$$

If $N^l = Out_k$:

$$\frac{\partial F^l_{\mathcal{G}}}{\partial o^l}(x,w,o^l(x,w)) = dF_{i_k}(G(x,w)) \quad (12)$$

If $N^l \notin Out$:

$$\frac{\partial F^l_{\mathcal{G}}}{\partial o^l}(x,w,o^l(x,w)) = \quad (13)$$
$$\sum_{N^j \in S(l)} \frac{\partial F^j_{\mathcal{G}}}{\partial o^j}(x,w,o^j(x,w)) dN^j_{r(l,j)}(I^j(x,w))$$

6.3 Complexity

Theorem 6.2 *With the hypothesis of theorem 6.1, we obtain that the time needed to compute the matricial operations required for calculating dF_G with the back-propagation method is proportionnal to :*

$$|O^F| \left[\sum_{N^l} |W^l| \left(2\,|O^l| - 1\right) + \sum_{N^l \notin Out} |O^l| \left(2 \sum_{N^j \in S(l)} |O^j| - 1 \right) \right] \tag{14}$$

The complexity of the direct algorithm applied to this modified model is simply the sum of the complexity obtained in theorem 5.1 with the time needed to compute the product between dG_w and dF (i.e.,a time proportionnal to $|O^F||W|(2|O| - 1)$).

6.4 Comparison

Once again, it is not possible to achieve a direct comparison between the computation times for both methods. But we can prove that if F is an error function (i.e., $|O^F| = 1$), the back-propagation is faster than the direct method, for a multilayer perceptron (i.e., we rediscover this classic result). Another important result is that computing with our generalized back-propagation the gradient of the error mode by a MLP represented with the mathematical language of our model is as fast as doing it with the classic back-propagation algorithm. In fact, the formulae are identical for both methods.

7 Conclusion

In this paper we have presented a general model for feedforward neural networks. On one hand, this model is powerfull enough to describe classic architectures such as multilayer perceptrons or wavelet networks. On the other hand, the model is simple enough to allow the derivation of an extended back-propagation algorithm. Unfortunately, this algorithm is no longer faster than the direct method, except for particular cases (such as the MLP). The obtained differentials can be used for gradient based neural learning.

The general model gives a formal framework and allows us to study overall properties of feedforward neural networks, such as differential computation. It has been used also to construct a simulator kernel which can handle arbitrary feedforward neural network in a very efficient way (see [2]).

References

[1] Léon Bottou and Patrick 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 Kauffman, 1991.

[2] Cédric Gégout, Bernard Girau, and Fabrice 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.

[3] Cédric Gégout, Bernard Girau, and Fabrice Rossi. A General Feed-Forward Neural Network Model. Technical report NC-TR-95-041, NeuroCOLT, Royal Holloway, University of London, May 1995. Available at `http://apiacoa.org/publications/1995/neurocolt1995.pdf`.