Geometrical Initialization, Parametrization and Control of Multilayer Perceptrons: Application to Function Approximation[0]

Fabrice ROSSI[1] and Cédric GEGOUT
Ecole Normale Supérieure de Paris
Thomson-CSF/SDC
7, rue des Mathurins, 92223 Bagneux Cedex, FRANCE

***Abstract*— This paper proposes a new method to reduce training time for neural nets used as function approximators. This method relies on a geometrical control of Multilayer Perceptrons (MLP). A geometrical initialization gives first better starting points for the learning process. A geometrical parametrization achieves then a more stable convergence. During the learning process, a dynamic geometrical control helps to avoid local minima. Finally, simulation results are presented, showing drastic reduction in training time and increase in convergence rate.**

I. Introduction

Mathematical theorems prove the existence of a two-layer perceptron, with sigmoidal units in the hidden layer, that approximate a given real valued multivariate continuous function to a given degree of accuracy [3, 5, 8]. Such approximation methods are very important for pratical purposes : they can be used, for instance, in black-box identification and control of nonlinear systems, or in time series prediction. The main problem is that the training of neural approximators is very difficult. Some authors have given initialization methods that reduce the training to a simpler problem (e.g. [4, 7]) but these methods depend on a selection method which extracts few key prototypes from an important data set, and furthermore, they introduce modifications to the simple MLP model. Decision tree based methods are also available (e.g. [2, 12]), but they are not really adapted for function approximation. Moreover, all these methods focus on the initialization process whereas all the approximation process must be improved : both initialization and learning.

In order to fulfill these requirements, we propose to switch from the standard MLP parametrization to a geometrical one, as proposed in [15]. The initialization of the network is carried out by a simplified version of the methods proposed in [10] and [14]. This new training process is faster than the classic one provided a good starting point is chosen among the possible ones. With the bad starting points, the network can get very fastly stuck into a false minimum. Many of these minima can be explained by the fact that some neurons give the same output on the training data and are thus useless. We present a simple geometrical method which allows us to control this behavior and to avoid many false minima. Used with the geometrical initialization and training methods, it yields very good results. Simulation results show that it greatly increases the convergence rate while reducing the average training time.

II. Geometrical parametrization

A. Geometrical point of view

The output of a two layer perceptron with one linear neuron in its last layer is given by :

$$f(\vec{x}) = \sum_{i=1}^{n} \gamma_i \sigma(\vec{\alpha}_i . \vec{x} + \beta_i) + \gamma_0, \tag{1}$$

where $\vec{x}$ is the vectorial input of the network, and σ is the transfer function of the hidden neurons. To speed up the training process, we propose to switch from the classic point of view to a more "geometrical" and intuitive one. We rewrite equation 1 in :

$$f(\vec{x}) = \sum_{i=1}^{n} \gamma_i \sigma\left(\vec{d_i} . (\vec{x} - \vec{t_i})\right) + \gamma_0 \tag{2}$$

We call $\vec{d_i}$ the dilatation parameter and $\vec{t_i}$ the translation parameter. As pointed out in [15], the partial derivative

$$\frac{\partial F}{\partial \vec{\alpha}_i} = \gamma_i \sigma'(\vec{\alpha}_i . \vec{x} + \beta_i)\vec{x}, \tag{3}$$

[0]In Proceedings of the IEEE International Conference on Neural Networks, Orlando, June 94.
Available at `http://apiacoa.org/publications/1994/icnn94.pdf`

[1]Up to date contact informations for Fabrice Rossi are avaible at `http://apiacoa.org/`

depends on the value $\vec{x}$, whereas the partial derivative

$$\frac{\partial F}{\partial \vec{d_i}} = \gamma_i \sigma' \left(\vec{d_i}.\,(\vec{x} - \vec{t_i}) \right) (\vec{x} - \vec{t_i}), \qquad (4)$$

depends on the value $(\vec{x} - \vec{t_i})$ and is thus less dependent from translation of all the input space and therefore more stable. Another reason for this choice is explained in section IV.

B. Complexity

Let $\mathcal{M}$ be a 2 layer perceptrons with n entries, p hidden neurons and one output, used with classic parametrization. Let C be the time needed to compute the product of two real numbers. If we assume that computing $\sigma(x)$ takes about $10C$, then the time needed to compute the ouput of $\mathcal{M}$ is about $p(2n + 12)C$ ([11]). The time needed to compute the gradient of the quadratic error made by $\mathcal{M}$ on one example is about $p(3n + 25)C$.

With the geometrical parametrization, new weigths are added to the network. In this case, the time needed to compute the output of $\mathcal{M}$ is about $p(3n + 11)C$. The computation time of the gradient is about $p(5n + 24)C$. Furthermore, the number of parameters is no longer $p(n+2)+1$ but $p(2n+1)+1$. Therefore, using this gradient is more time consuming than using the classic gradient.

Working with a small n is not really a problem and the complexity difference can be neglected. But if the entries number is large, the computation time of the geometrical method is about 1.7 times longer than the one of the classic method.

III. Geometrical initialization

The equation $\vec{\alpha_i}.\vec{x} + \beta_i = 0$ describes a hyperplane. Due to the general properties of the commonly used transfer functions, the output of the i^{th} neuron is non constant only near this hyperplane. As pointed out in [14], the relative positions of the hyperplane and the input data are very important. A bad configuration may result in a local minima. It is very difficult indeed for a gradient procedure to correct a constant output on the training set because the derivative of the transfer function is localized and non zero only near the hyperplane. Therefore, if this hyperplane is "far away" from the input data, the partial derivatives of the error $\frac{\partial E}{\partial \alpha_{ij}}$ are close to zero, which slows down any learning method that use the gradient.

The classic random initialization does not enable to find a good starting point for the training process. Indeed, picking randomly and independently $\vec{a_i}$ and β_i does not enable to control the position of the hyperplane because this position depends on $\frac{-\beta_i}{||\vec{a_i}||}$. On the contrary, the proposed parametrization turns the simple random initialization into a more efficient method : one can achieve easily a simple control on the hyperplane position because it depends rather independently on $\vec{t_i}$ and $\vec{d_i}$.
In fact, the ouput of a neuron depends on three parameters :

1. The position of its "center of activity" (i.e., the translation parameter $\vec{t_i}$).
2. The orientation of its decision hyperplanes (i.e., $\frac{\vec{d_i}}{||\vec{d_i}||}$).
3. The selectivity of its transfer function (i.e., $||\vec{d_i}||$).

Obviously, a simple way to achieve better performances than with a simple random initialization is to separate the initialization process into three parts, each of them dealing with one parameter.
A simple version of this initialization process was used :

The three parameters are selected randomly from three differents sets. The translation set is, for instance, the part of the input space on which the function to learn is defined. The direction set can be the unit sphere and the selectivity set is chosen in order to avoid saturation or linearity of the starting neurons on all the training examples.

Let us consider, for instance, the case of an one input MLP. If the training examples are distributed in the interval $[a, b]$, then we take for the translation set $[a, b]$, for the orientation set $\{-1, 1\}$ and for the selectivity set $[\frac{\lambda_1}{b-a}, \frac{\lambda_2}{b-a}]$, where λ_i are properly selected constant depending on the transfer function. For example, if $\sigma(x) = \tanh(\frac{x}{2})$ is used, saturation of the output begins when $|x| > 5$. The output can be well approximated by $\frac{x}{2}$ when $|x| < 1$. In order to avoid linearity or excessive saturation, one can set for instance $\lambda_1 = 1$ and $\lambda_2 = 10$.

The weights of the last layer are not really important because the error is a quadratic function of them and can thus be optimized by any elaborate gradient method. A good method is to initialize these weights with small values or to set them to zero.

With this low cost method, we obtain a rather good control on the position of the hyperplanes and of their selectivity. On the contrary, the classic parametrization used with such a simple method does not allow any control on the position of the hyperplanes, as explained before. We can yet use the geometrical parametrization for initialization and switch back to the classic modal before learning.

Of course, more powerful methods can be used (see

[6, 14, 15]). But this one is general and its efficiency does not depend on the data.

IV. Useless neurons

A. Description of the problem

During training process, one can frequently obtain useless neurons, that is a neuron which gives an output close to another neuron :
Let us assume that a classic transfer function is used. The function σ must fulfill the following conditions :

- σ is odd.
- $\lim_{x\to\infty} \sigma(x) = 1$.
- $\sigma'(0) = \kappa > 0$.

Let us consider two neurons giving a summed output,

$$F(\vec{x}) = \gamma_1\sigma(\vec{d_1}.(\vec{x} - \vec{t_1})) + \gamma_2\sigma(\vec{d_2}.(\vec{x} - \vec{t_2})).$$

Because of the oddity of σ, we can assume $\gamma_i > 0$. If the two decision hyperplanes defined by the neurons are close, we have :

$$\forall \vec{x},\ \left| \frac{\vec{d_1}}{||\vec{d_1}||}.(\vec{x} - \vec{t_1}) \right| \simeq \left| \frac{\vec{d_2}}{||\vec{d_2}||}.(\vec{x} - \vec{t_2}) \right|$$

If we have further $\vec{d_1}.\vec{d_2} > 0$ and $||\vec{d_1}|| \simeq ||\vec{d_2}||$, then F can be approximated by only one neuron, i.e., $F(\vec{x}) \simeq \gamma_m\sigma(\vec{d_m}.(\vec{x} - t_m\vec{d_m}))$, where $\vec{d_m} = \frac{\vec{d_1}+\vec{d_2}}{2}$, $\gamma_m = \gamma_1 + \gamma_2$ and $t_m = \frac{\vec{t_1}.\vec{d_1}+\vec{t_2}.\vec{d_2}}{2}$. One of the two neurons is thus useless.

The main problem is that no gradient method using classic parametrization may separate the two neurons. Since the hyperplanes are close, $\vec{\alpha_i}.\vec{x} + \beta_i$ are close on the training set. Therefore the partial derivatives calculated using equation 3 are close too and the two neurons are moved together by a gradient descent algorithm. On the contrary, the closeness of the hyperplanes does not imply that $\vec{t_1}$ and $\vec{t_2}$ are nearly equal. The partial derivatives calculated using equation 4 can thus be different. The geometrical parametrization is therefore less sensitive to such a collapse of neurons.

B. Dynamic control

Let us first consider the case of a function from $I\!R$ to $I\!R$. The sum of the output of two neurons is : $F(x) = \gamma_1\sigma(d_1(x - t_1)) + \gamma_2\sigma(d_2(x - t_2))$. We assume that t_1 and t_2 are nearly equal and we want to suppress one neuron. One simple way to do that is to train an unique neuron to approximate the output of the two others. This might be efficient but rather slow. One can use also some pruning method (e.g. [9]). We prefer a simple direct geometric method :
The slope of $F(x)$ for $x \simeq 0$ is $\kappa(\gamma_1 d_1 + \gamma_2 d_2)$. If $|\gamma_1| > |\gamma_2|$, then the first neuron is the dominating one. We will assume that the second neuron is useless if the slope of the sum has the same sign as the slope of the dominating neuron. If this condition is fulfilled, the neuron given by $\gamma_m\sigma(d_m(x - t_m))$ gives a “good” approximation of the two former neurons, if $\gamma_m = \gamma_1 + \gamma_2$, $t_m = \frac{t_1+t_2}{2}$ and $d_m = \frac{\gamma_1 d_1 + \gamma_2 d_2}{\gamma_m}$. This allows us to use the remaining neuron to reduce the error on the rest of the training set (see the simulation part).

This heuristic can be extended to the multidimensional case as follows :

We first assume that the hyperplanes defined by the neurons are close. Let $\vec{n_i}$ be $\frac{\vec{d_i}}{||\vec{d_i}||}$. We assume that either $\vec{n_1}$ and $\vec{n_2}$, either $\vec{n_1}$ and $-\vec{n_2}$ are close, i.e., $|\vec{n_1}.\vec{n_2}| \simeq 1$. Let $\vec{n_m}$ be $\frac{1}{2}(\vec{n_1} + \epsilon\vec{n_2})$ (with $\epsilon = 1$ or $\epsilon = -1$, in order to sum the close vectors). Let θ_i be $\vec{t_i}.\vec{n_i}$, and let x be the projection of $\vec{x}$ on $\vec{n_m}$. The summed output of the two neurons is nearly equal to : $\gamma_1\sigma(||\vec{d_1}||(x - \theta_1)) + \epsilon\gamma_2\sigma(||\vec{d_2}||(x - \theta_2))$.

The real function method can be applied to this projected equation. We obtain a single neuron whose translation parameter and dilatation parameter are respectively $t_m = \frac{\theta_1+\theta_2}{2}$ and $d_m = \frac{\gamma_1||\vec{d_1}|| + \epsilon\gamma_2||\vec{d_2}||}{\gamma_m}$, with $\gamma_m = \gamma_1 + \epsilon\gamma_2$. In the original space, we take $\vec{d_m} = d_m\vec{n_m}$ and $\vec{t_m} = t_m\vec{n_m}$. Of course γ_m remains unchanged.

C. Complexity

In order to find out useless neurons, a distance between every pair of hyperplanes must be computed. The time needed to compute one distance is about $6nC$, if $||\vec{d_i}|| = 1$. The time required to compute the $\vec{n_i} = \frac{\vec{d_i}}{||\vec{d_i}||}$ is about $p(3n + 12)C$. Therefore, the total time is about $p(3np + 12)C$. If we assume that the probability of useless neurons is rather low, we can neglect the time needed to compute the parameters of the new neurons.

With an off-line learning, such as the Broyden-Fletcher-Golfard-Shanno (BFGS) conjugate gradient ([13]), the time needed to complete an iteration is about $(4n + 27 + 10(2n + 12))pkC$ where k is the number of learning patterns (we assume here that about 10 evaluations of the error are needed to perform the one-dimensional minimization required by the BFGS method). The ratio is therefore less than $\frac{p(3n+12)}{k(24n+147)}$. It is well known that only $k - 1$ hidden units are needed to learn exactly an arbitrary mapping defined on k examples [1]. The ratio is thus less than $\frac{1}{8}$. Therefore, the time needed to check for useless neurons can be neglected if this method is applied only every 10 iterations.

V. Simulation results

This section compares the performances of the several proposed optimizations on a simple example :

The function to learn was generated by a 2 layer perceptron, with 4 hidden neurons. 31 training points were uniformly spread throughout the training interval, $[-3, 4]$ (see figure 1). The BFGS conjugate gradient algorithm was used to train 2 layer perceptrons. They had between 4 and 8 hidden neurons. The learning stopped when the mean square error on the training set was lower than 0.01 or when the number of iterations exceeded 500.

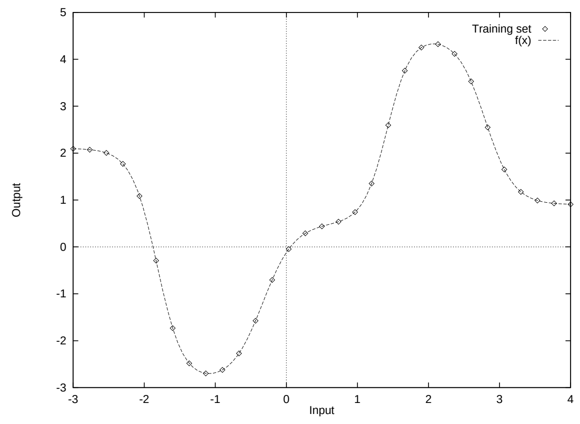

Figure 1: Function to learn

Two initialization methods were used : the classic one, called ClaI (i.e., weigths are chosen randomly from $[-\frac{2}{3}, \frac{2}{3}]$), and the geometrical one, called GeoI (the translation and slope parameters are chosen randomly respectivly from $[-3, 4]$ and from $[3, 10]$).

Two parametrization methods were used : the classic one, called ClaP, and the geometrical one, called GeoP.

We have used a control method to avoid useless neurons : when the mean square error remains almost constant during 10 iterations, the presence of useless neurons is checked. When the distance between hyperplanes is inferior to 8% of the training set length (7 here), the two concerned neurons are approximated by only one. Let ρ be the normalized mean square error defined by :

$$\rho(\vec{x}_i) = \frac{(F(\vec{x}_i) - y_i)^2}{\sum_j (F(\vec{x}_j) - y_j)^2},$$

where F is the MLP calculated function and y_i the desired output for input $\vec{x}_i$. Then the new translation parameter for the remaining neuron is defined as the center of gravity of the training set :

$$\vec{t} = \sum_j \vec{x}_j \rho(\vec{x}_j)$$

The dilatation parameter of this neuron keeps its old value and γ_i is set to zero.

In order to obtain representative statistical data, the training was repeated 100 times for each initialization method, each time starting with a different set of network parameters.

Table 1 gives the percentage of convergent runs and the average number of iterations before convergence for the convergent runs. The time needed to compute an iteration of the algorithm is directly proportionnal to the number of hidden neurons. The number called "relative iterations" is the product of the average iteration number and the complexity ratio between the 6 or 8 neurons case and the 4 neurons case. The obtained averages are thus comparable.

The geometrical approach results in a drastic reduction of the training time. But without dynamic control, the number of non convergent runs increases. For 8 hidden neurons, the convergence rate is only 93% and the simple classic method is thus more reliable.

The dynamic control increases the convergence rate for the geometrical algorithms but has a rather bad influence on the classic method. The best algorithm is obviously the total geometrical algorithm, used with a dynamic control. This fact is confirmed by other simulations.

VI. Conclusion

In this paper, we have presented a geometrical approach of neural network learning. This method relies on previous works and on new ideas such as the important dynamic control. It can divide by four or more the training time and can increase the percentage of convergent runs. The proposed approach must now be validated on more data, especially on multi-dimensional data and real-world problems, such as time series prediction. In addition, future work will also include development of elaborated initialization processes and control methods, and comparison between transfer functions.

References

[1] Masahiko Arai. Mapping abilities of three-layer neural netwoks. In *Proc. Int. Joint Conf. Neural Networks*, volume I, pages 419–423, 1989.

[2] Krzysztof J. Cios and Ning Liu. A machine learning method for generation of a neural net-

Hidden Neurons	4			6			8		
Parametrization	ClaP	ClaP	GeoP	ClaP	ClaP	GeoP	ClaP	ClaP	GeoP
Initialization	ClaI	GeoI	GeoI	ClaI	GeoI	GeoI	ClaI	GeoI	GeoI
Without control	26 %	34 %	37 %	92 %	85 %	88 %	99 %	93 %	93 %
iterations	334	112	61.2	231	69.4	46.5	179	55.5	26.2
relative iterations				346	104	69.8	358	111	52.4
With control	35 %	47 %	85 %	72 %	95 %	100 %	85 %	99 %	100 %
iterations	282	89.3	69.9	263	66.1	38.5	224	55.7	28.8
relative iterations				395	99.1	57.8	448	111	57.6

Table 1: Simulation results : convergence rates and average numbers of iterations

work architecture : A continuous ID3 algorithm. *IEEE Trans. On Neural Networks*, 3(2):280–291, March 1992.

[3] G. Cybenko. Approximation by Superposition of a Sigmoidal Function. *Mathematics of Control, Signals and Systems*, 2:303–314, 1989.

[4] Thierry Denoeux and Régis Lengellé. Initializing Back Propagation Networks With Prototypes. *Neural Networks*, 6:351–363, 1993.

[5] Ken-Ichi Funahashi. On the approximate realization of continuous mappings by neural networks. *Neural Networks*, 2:183–192, 1989.

[6] Cédric Gégout and Fabrice Rossi. Continuous Parameter Optimization with Genetic Algorithms. Application to Neural Network Initialization. Technical report, Thomson-CSF/SDC, Nov. 1993.

[7] Shlomo Geva and Joaquin Sitte. A constructive method for multivariate function approximation by multilayer perceptrons. *IEEE Trans. On Neural Networks*, 3(4):624, July 1992.

[8] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.

[9] Ehud D. Karnin. A Simple Procedure for Pruning Back-Propagation Trained Networks. *IEEE Trans. On Neural Networks*, 1(2):239–242, June 1990.

[10] D. Nguyen and Widrow B. Improving the Learning Speed of 2-layer Neural Networks by Choosing Initial Values of the Adaptive Weights. In *Proc. Int. Joint Conf. on Neural Networks*, volume 3, pages 21–26, San Diego, 1990. IEEE.

[11] Fabrice Rossi and Cédric Gégout. Apprentissage pour les mlp : étude algorithmique. Technical report, Thomson-CSF/SDC, Oct. 1993.

[12] Ishwar K. Sethi. Entropy nets : From decision trees to neural networks. *Proc. IEEE*, 78(10):1605–1613, October 1990.

[13] D. F. Shanno. Conjugate Gradient Methods with Inexact Searches. *Mathematics of Operations Research*, 3(3):244–256, 1978.

[14] Lodewyk F. A. Wessels and Etienne Barnard. Avoiding false local minima by proper initialization of connections. *IEEE Trans. On Neural Networks*, 3(6):899–905, November 1992.

[15] Qinghua Zhang and Albert Benveniste. Wavelet networks. *IEEE Trans. On Neural Networks*, 3(6):889–898, November 1992.