ESANN'1995 proceedings - European Symposium on Artificial Neural Networks
Brussels (Belgium), 19-20-21 April 1995, D-Facto public., ISBN 2-9600049-3-0, pp. 365-371

An evolutive architecture coupled with optimal perceptron learning for classification

J-M. Torres Moreno, Pierre Peretto and Mirta B. Gordon.

CEA/Département de Recherche Fondamentale sur la Matière Condensée
CEN-G - 17, avenue des Martyrs - 38054 Grenoble Cedex 9 - France

Abstract. We present a new incremental learning algorithm that generates small networks with very good generalization performances. It combines a strategy for generating internal representations with a highly performant perceptron training algorithm, Minimerror. Results of tests on artificial and real classification problems are reported.

I. Introduction

Learning binary classifications from examples is one of the main challenges for neural networks. Given a learning set of examples, the aim is to design and train a network that will correctly classify new inputs, i.e. with high generalization performance. Although a feedforward perceptron with a single hidden layer is sufficient to approximate a given function, the number of units needed to reach a given accuracy is not yet known. Thus, a learning rule for networks with hidden units should deal with the number of hidden units, the internal representations (IR) of the input patterns, i.e. the states of the hidden units to be associated to each input of the training set, and the values of the synaptic weights all at the same time.

Several simplifications make the problem more tractable. The simplest one consists in freezing the number of hidden units, and to minimize the number errors. If hidden and output units are *continuous* variables, the function to minimize is derivable, and the weights are determined with a gradient descent. This is what Backpropagation (BP) does. The IR's are a by-product of the algorithm. One of the drawbacks of BP is that some hidden units may be working in the linear regime, and might be replaced by less non-linear units. Pruning techniques cope with this problem, but learning has to be reiterated.

Learning in networks with *binary* hidden units has to manage with the IR's. This approach is very attractive, because once a strategy to generate the IR is proposed, it reduces the problem to that of learning with single perceptrons (as many as hidden and output units are needed). The strategies proposed so far for learning with a fixed number of binary hidden units[1,2] are not guaranteed to converge to a solution. Evolutive learning strategies are more promising, because they handle both, the number of hidden units and the synaptic weights at the same time, to learn the training set. Incremental procedures, that add hidden units one after the other, generate mainly two classes of architectures : tree-like networks[3,4,5,6,7,8,9] or multilayered perceptrons with each hidden layer fully connected to the preceding one[10,11,12,13,14]. We are interested in the latter, because tree-like networks seem less robust : the failure of a single branch may disconnect a whole sub-network from the input.

In this paper, we present an incremental algorithm that generates a network with only *one* hidden layer - we suggest to call it *Monoplane* - by combining the Tiling-like/Offset strategies[12,14] and a suggestion by Frean[7]. We use *Minimerror*[15,16,17] to train the individual units, a learning algorithm for binary perceptrons that finds a solution whether the training set is separable or not : it minimizes the number of training errors if the training set is non-separable, and it achieves optimal generalization if the training set is linearly separable[18]. Its good performance is crucial in order to generate small networks, believed to generalize better than large ones. The algorithm is presented in the next section. Its implementation, and results of tests on artificial problems are presented in §III. Its application to a real world benchmark is presented in §IV. Section V is devoted to the conclusions.

II. The evolutive learning algorithm

Consider a training set of P input-output pairs $\left\{(\vec{\xi}^{\mu},\tau^{\mu})\right\}$, μ=1,2,...,P. The inputs $\vec{\xi}^{\mu}=(1,\xi_1^{\mu},\xi_2^{\mu},\ldots,\xi_N^{\mu})$ are real valued N+1 dimensional vectors, with the first component acting as a bias. The outputs are binary $\tau^{\mu}=\pm 1$. We want to construct a neural network with one hidden layer connected to the N+1 input units, and one output neuron connected to the hidden neurons. During training, the size of the hidden layer grows until the number of training errors vanishes. After learning, the hidden neurons h, $1\le h\le H$, have synaptic weights w_{hi}, $0\le i\le N$, the w_{h0} are biases. The output has a bias w_0 and and synaptic weights w_h. When the input is pattern $\vec{\xi}^{\mu}$, the states $\vec{\sigma}^{\mu}=(1,\sigma_1^{\mu},\ldots,\sigma_H^{\mu})$ of the hidden neurons, and the network's output σ_{out}^{μ} are

$$\begin{cases}\sigma_h^{\mu}=\mathrm{sign}\left(\sum\limits_{i=0}^{N} w_{hi}\xi_i^{\mu}\right) & \text{for } 1\le h\le H \\ \sigma_{out}^{\mu}=\mathrm{sign}\left(\sum\limits_{h=0}^{H} w_h\sigma_h^{\mu}\right) & \end{cases}\qquad (1)$$

The states σ_h^{μ} define the *internal representations* of the inputs, which have to be *faithful*[11] : two input states whose outputs are different should have different IR's. As our network has only *one* hidden layer, these *internal representations* have also to be *linearly separable.*

The learning algorithm has two stages. The first one is identical to the first hidden layer construction of the Offset algorithm[12] and of the Tiling-like Learning in the Parity Machine[14]. It generates faithful IR's, such as the network's output is the parity of the IR. However, contrary to Offset, that implements the parity with a second hidden layer which has to be subsequently pruned, we go on adding hidden units following a variation to the Upstart algorithm suggested by Frean[7], until the IR's become separable. The advantage of this strategy with respect to Offset is that it avoids the second layer pruning and, in general, it ends up with networks having a lower number of synaptic weights. We start by connecting a first hidden neuron (h=1) to the input units, that learns the training set. If the number of training errors vanishes, $\varepsilon_t(h=1)=0$, the algorithm stops ; the training set is linearly separable and the network is a perceptron. If $\varepsilon_t(1)\neq 0$, this unit is the first one of the hidden layer. The algorithm proceeds as follows :

Stage 1 : If $\varepsilon_t(h)\neq 0$, a new neuron is connected to the input, $h \leftarrow h+1$, and is given to learn $\tau_h^\mu = \tau_{h-1}^\mu \sigma_{h-1}^\mu$, $(1 \le \mu \le P)$. We go on adding hidden units until one gives $\varepsilon_t(h)=0$. A solution exists [12, [14] for each added unit that decreases by at least 1 the number of errors of the previous one, thus ensuring convergence of this stage. Once the condition $\varepsilon_t(h)=0$ is met, go to stage 2.

Stage 2 : connect the output unit to the hidden units, and use the following training set : $\left\{(\vec{\sigma}^\mu, \tau^\mu)\right\}$ for $1 \le \mu \le P$, where $\vec{\sigma}^\mu = (1, \sigma_1^\mu, \ldots, \sigma_h^\mu)$ are the outputs of the hidden units. If error-free learning is achieved, the IR's are linearly separable, and the algorithm stops : the hidden layer has H=h units. Otherwise, we go back to stage 1 to add new hidden units, with $h \leftarrow h+1$ and $\tau_h^\mu = \tau^\mu \sigma_{out}^\mu$, where σ_{out}^μ is the output of the output neuron to pattern μ. Here again, this back and forth procedure converges : a solution exists that decreases the number of errors of the output neuron by at least 1 each time. In practice, a single return to stage 1 was enough to find a solution in most of our simulations.

The final number of hidden units depends on the performance of the learning algorithm used to train the individual perceptrons. Most incremental strategies use the Pocket algorithm[5] ; it has no natural stopping condition, which is left to the patience of the trainer. None of the proposed improvements[19,20] are guaranteed to find the *best* solution, which endows the perceptron with the lowest generalization error if the training set is linearly separable, and minimizes the number of errors otherwise. Minimerror[15,16,17,18] is a learning algorithm recently developed to this end, that has been shown theoretically[16] and numerically[17] to fulfil both conditions. It generates normalized weights $\vec{w}$: $\vec{w} \bullet \vec{w} = \sum_{j=0}^{M} (w_j)^2 = M+1$ (M=N for the hidden units, and M=H for the output unit) that minimize a continuous cost function representing a noisy measure, at a temperature T, of the training errors

$$E = \frac{1}{2} \sum_{\mu=1}^{P} \left[1 - \tanh \frac{\gamma^\mu}{2T} \right] \tag{2}$$

The *stability* or *margin* of pattern μ, $\gamma^\mu \equiv \frac{\tau^\mu \vec{w} \bullet \vec{\xi}^\mu}{\sqrt{N+1}}$, measures the distance of pattern μ to the separating hyperplane, with positive sign if it is well classified, negative otherwise. Minimization of (2) is achieved through deterministic annealing, which is nothing else than a gradient descent combined with a slow decrease of the temperatureT through $1/(T-\delta T) = 1/T + \delta(1/T)$, where $\delta(1/T)$ is the annealing rate parameter. A full description of the implementation of Minimerror will be presented elsewhere[21].

ESANN'1995 proceedings - European Symposium on Artificial Neural Networks
Brussels (Belgium), 19-20-21 April 1995, D-Facto public., ISBN 2-9600049-3-0, pp. 365-371

III. Implementation and preliminary tests

The implementation of stages 1 and 2 of Monoplane is straightforward. There are two kinds of adjustable parameters : those of Minimerror and the initialization of weights. In Minimerror, the initial temperature, the asymmetry and the learning strength were kept constant, with the values given in[16] in all our simulations. The effect of modifying the annealing rate is discussed in IV. Initialization of weights may be random, like in BP. However, better results are obtained with the following initial weights : the first hidden neuron is initialized with Hebb's rule :

$$w_{1i}^{\text{init}} = \frac{\sqrt{N+1}}{|\vec{w}|} \sum_{\mu=1}^{P} \xi_i^\mu \tau^\mu \ ; \ |\vec{w}|^2 = \sum_{i=0}^{N} \left(w_{1i}^{\text{init}}\right)^2 \tag{3}$$

where the pre-factor $\sqrt{N+1} \, / \, |\vec{w}|$ ensures the proper normalization. Subsequent hidden neurons are initialized with the solution of the preceding ones. The output unit is initialized with Hebb's rule, calculated like in (3), but with H at the place of N and the IR's σ_h^μ instead of the input patterns ξ_i^μ. With this initialization, Monoplane is completely deterministic.

We tested Monoplane on the exhaustive learning of the parity of N binary inputs, for $1 \le N \le 11$ (for N=11 the training set has P=2048 patterns). This well known problem confirmed that Monoplane generates the smallest and most robust network, which has H=N and maximizes the lowest stabilities of learned patterns, on the hidden units as well as on the output unit. . To our knowledge, no other learning algorithm was yet reported to produce this result for the largest value of N tested here.

Minimerror was devised for binary inputs, located at the vertex of the centred hyper-cube of side length 2. Its performance on problems where the states of input neurons are real numbers was tested on the separation of two intertwined spirals. A very simple pre-processing of the input data that bring the patterns back *inside* the hyper cube was found to give good results. Thus, for real valued inputs, the following linear transformation has to be applied to each component of each pattern :

$$\xi_i^\mu \leftarrow 2\frac{\xi_i^\mu}{M-m} - 1 \tag{4}$$

where $M \equiv \underset{i,\mu}{\text{Max}}\left\{\xi_i^\mu\right\}$ is the maximum among all the components of the training set $\left\{\xi_i^\mu\right\}$, and $m \equiv \underset{i,\mu}{\min}\left\{\xi_i^\mu\right\}$ is the minimum.

IV. Application to the classification of sonar targets.

We tested Monoplane on the classification problem of sonar targets, studied by Gorman et al. with BP[22]. Input patterns are 128 sonar returns collected from either a metal cylinder or a cylindrically shaped rock. They are coded with 60 real numbers in the interval [0,1]. We pre-processed the inputs with (4) to bring them back to the binary hyper-cube. We studied both aspect-angle independent (AAI) and aspect-angle dependent (AAD) data classification[22].

Consider the AAD problem first. A subset of 104 representative patterns was selected as training set, the remaining 104 patterns were left to test the generalization performance[22]. The results of Monoplane with an annealing schedule for Minimerror of $\delta(1/T) = 0.001$, are summarized in Table 1, together with BP's results. Because Monoplane is a deterministic algorithm there is no dispersion in our results. Surprisingly, the generalization performance is worse with two hidden units than with a single perceptron (i.e. with only the first hidden unit connected), a phenomenon usually interpreted as overfitting. To understand this result, we trained the first hidden unit with an extremely slow annealing schedule ($\delta(1/T) = 10^{-4}$). We found that both, the training set and the testing set of this benchmark are linearly separable, but with different hyperplanes. Thus, neither the training set nor the testing set are representative enough of the problem. By including in the training set the 17 testing patterns that our 2-hidden network could not classify correctly, Monoplane (with the initial value $\delta(1/T) = 0.001$ for Minimerror) generated a 2-hidden units network having 100% of learning *and* generalization performance.

The AAI problem uses a different partition of the same patterns to create the training and testing sets. The testing set contains 16 randomly selected patterns among the 208 available ones. The 192 remaining patterns constitute the training set. Results corresponding to 9 different testing sets are reported on Table 2. Depending on the learning set, Monoplane ended with 2, 3 or 4 hidden units. In every case, when including the wrongly classified patterns of the testing set into the training set, 100% performance on training and generalization is obtained.

These results shed light on the ubiquitous problem of overfitting. In this particular problem, seemingly imperfect generalization may be an indication of insufficient amount of data, rather than overfitting.

H	ε_t(MP)	ε_t(BP)	ε_g(MP)	ε_g(BP)
0	3	21.5±0.9	16	27.9±1.8
2	0	3.9±0.09	17	14.8±1.1
3	-	1.9±0.02	-	12.9±0.4
6	-	0.6±0.01		11.1±0.3

Table 2. AAD series.
ε_t=number of training errors
ε_g=generalization errors.
BP=results of BP[22]
MP=results with Monoplane.

H	ε_t(MP)	ε_t(BP)	ε_g(MP)	ε_g(BP)
0	5.2±0.2	20.4±0.5	3.9±0.3	3.7±0.4
2	0	6.7±0.04	5.5±0.2	2.9±0.2
3	0	2.3±0.01	3.6±0.2	2.9±0.3
4	0		2.9±0.2	
6	-	0.6±0.01		2.7±0.2

Table 1. AAI series.
ε_t=number of training errors
ε_g=generalization errors.
BP=results of BP[22]
MP=results with Monoplane.

IV. Conclusion

We presented a new incremental learning algorithm, Monoplane, that generates *small* networks with very good *generalization* performances. These properties stem from the combination of a good strategy for generating internal representations, coupled to a highly performant perceptron training algorithm. We presented tests on artificial and real classification problems. Monoplane proved to be a powerful tool for efficient learning and for data analysis. The results presented in this paper suggest several directions for further study. Investigation of overfitting in other problems, implementation of on line learning by including wrongly classified patterns in the training set, extensions to overlapping and noisy patterns distributions are topics currently in progress.

References

[1] Mitchison, G.J. and Durbin, R.M., Biol Cybern **60** (1989) 345.
[2] Nabutovsky D, Grossman T. and Domany E, Complex Systems **4**(1990) 519.
[3] Nadal, J.P., J.Phys.A (Math.Gen.) **22** (1989) 2191.
[4] Sirat, J.A. and Nadal, J.P., Network **1** (1990) 423.
[5] Gallant, S.I., IEEE Transactions on Neural Networks **1** (1990) 179.
[6] Fahlman S.E. and Lebiere C., in: Advances in Neural Information Processing Systems 2, D.S. Touretzky ed. (Morgan Kaufmann, San Mateo, CA., 1990) p.574.
[7] Frean, M., Neural Computation **2** (1990) 198.
[8] Golea, M. and Marchand, M., Europhys. Lett. **12** (1990) 205.
[9] Gray, D.L., IEEE Transactions on Neural Networks **3** (1992) 176.
[10] Rujan, P. and Marchand, M., Complex Systems **3** (1989) 229.
[11] Mézard, M. and Nadal, J.-P., J.Phys.A **22** (1989) 2191.
[12] Martinez, D. and Esteve, D., Europhys. Lett. **18** (1992) 95.
[13] Knerr, S., Personnaz, L. and Dreyfus, G., in: Neurocomputing : Algorithms, Architectures and Applications, J.H. F. Fogelman ed. (Springer, 1990) .
[14] Biehl, M. and Opper, M., Phys. Rev. **A 44** (1991) 6888.
[15] Gordon, M.B., Peretto, P. and Berchier, D., J.Phys.I France **3** (1993) 377.
[16] Gordon M.B. and Berchier D, in ESANN'93, M. Verleysen ed. (D facto, Brussels, 1993) p.105.
[17] Raffin, B. and Gordon, M.B., Submitted (1994) .
[18] Gordon, M. and Grempel, D., Europhys. Lett. **To be published** (1994) .
[19] Frean, M., Neural Computation **4** (1992) .
[20] Moreno, J.M., Castillo, F. and Cabestany, J., ESANN'93 (1993) 33.
[21] Torres-Moreno, J.M. and Gordon, M.B., In preparation (1994) .
[22] Gorman, R.P. and Sejnowski, T., Neural Networks **1** (1988) 75.