NEUROINTERFACES: LEARNING BY GENETIC ALGORITHMS

Marcelo Malini Lamego [1] **Edson de Paula Ferreira** [2]
Bernard Widrow [3]

Stanford University, E.E. Dept., CA 94305-9510 USA,
mmlamego@stanford.edu

Abstract: This article aims at showing how Genetic Algorithm (GA) and Vector Quantization (VQ) can be applied to the training of neurointerfaces. The GA in combination with a vector-quantized neuron model are used to train a neurointerface that helps a human operator to back up a scale model truck connected in a single-trailer configuration.

Keywords: Neural Networks, Genetic Algorithms, Vector Quantization, Inverse Control, Adaptive Systems

1. INTRODUCTION

For many tasks, productivity, safety, and liability conditions require a considerable degree of skill from human operators. In order to overcome lack of skill, special human-machine neurointerfaces (Widrow *et al.*, 1998) may be adopted. The basic idea is to change the operational space through a neural network, allowing the human operator to interact with the process through less-specialized commands. Accordingly, the operator devotes his attention to solve a less complex problem, directly at the task level. The objective is to improve the productivity and safety levels of such tasks even in the case of unskilled operators.

Typically, the design of neurointerfaces involves the training of recurrent neural networks. This comprises the search for a set of parameters (weights) that are a solution for a predefined objective function (mean-square error). Due to the non-convex nature of the objective function, a considerable number of local minima may exist. Accordingly, the chances of a search algorithm based on gradient techniques getting stuck in local minima (with unacceptable objective function values) are significant.

Genetic Algorithms (GA) (Holland, 1975) provide an interesting alternative for neural network training. GA is an iterative procedure which maintains a set of candidate solutions (populations). New populations are generated by means of a randomized "selection procedure" through idealized genetic recombination operators (reproduction, crossover and mutation). These features greatly reduce the occurrence of local-minimum solutions during neural network training.

Montana and Davis (1989) proposed the first approach to evolve weights in a neural network using GA. That is, they were using the GA instead of backpropagation (Rumelhart *et al.*, 1986) as a way of finding a good set of weights that was a solution for a neural network approximation problem.

The computational effort to run a GA is usually greater than the one required for a conventional gradient-based method such as backpropagation. Typical neural network applications require a great number of parameters to be adjusted. Due to the discrete nature of genetic algorithms, every weight in the neural network has to be quantized

---

[1] Ph.D. student sponsored by UFES/CNPq, Brazil
[2] Visiting Professor sponsored by UFES/CAPES, Brazil
[3] Professor

and the resulting values represented by binary strings (genes). These strings are concatenated and form all the possible structures of a population (chromosomes). The chromosome length is a function of the number of weights and the adopted weight quantization level. For instance, in a 100-weight neural network topology, if an 8-bit representation is used for each weight, the resulting structures will have a length of $100 \times 8 = 800$ bits. Normally, a population of the same order as the chromosome length is used to prevent "small individual variety", cause of local-minimum solutions. Since a simulation is required for each individual in the population, in order to evaluate its relative performance (fitness), the required computational power may be prohibitive even for such a small network.

In order to overcome the problem of fast growing GA chromosome lengths in neural network training applications and, indeed, to reduce the computational complexity of GA runs, a more rigorous approach can be adopted for the weight quantization procedure. Particularly, Vector Quantization (Gersho and Gray, 1992) can be used to fully exploit the statistical dependence among coefficients of a neuron. A single Vector Quantization process can be used to quantize all the weights of a neuron, instead of scalar quantization for each neuron weight. Here, the neuron's weight "codebook" is obtained by only assuming a few constraints on its inputs and output signals.

This paper is divided in 4 sections. Section 2 introduces a quantized model for a neuron. Section 3 discusses the design of neurointerfaces. Section 4 gives experimental results of GA and vector-quantized neurons applied to a trainable neurointerface application, the truck backer.

2. THE QUANTIZED MODEL OF A NEURON

Vector Quantization (VQ) in its simplest form observers a real vector variable in a continuous range of possible amplitudes and selects the corresponding nearest approximating values from a predefined finite set of allowed numerical values (the codebook). This idea will be used to greatly reduce the number of bits required to represent the weights, making GA optimization feasible.

The process of Vector Quantization maps the set of real n-vectors, $\mathbf{R}^n$, to a discrete output set. More precisely, an N-point vector quantizer of dimension n, here denoted by the capital letter $\mathbf{Q}$, can be defined as a mapping where $\mathbf{Q} : \mathbf{R}^n \rightarrow C$ is the output set or codebook defined as

$$C \equiv \{a_1, a_2, \cdots, a_N\} \subset \mathbf{R}^n \qquad (1)$$

The output values a_i are the reproduction values (also designated as output points or output levels) of the variables being quantized and N represents the codebook size $|C| = N$.

In this paper, we are concerned with the process of weight quantization of neurons. Therefore, the weight vector of a neuron is quantized to form a discrete variable, which can assume only a finite number of real values defined by a particular codebook. The inputs and output of each neuron are continuous variables in predefined continuous ranges of possible amplitudes.

VQ is usually, but not exclusively, used for the purpose of data compression involving digital signals. Normally, the statistical data dependencies among the elements of a signal vector are either known or obtained through the preprocessing of acquired data. This leads to a straightforward application of nearest neighbor encoding algorithms for the construction of vector quantizer codebooks.

Unfortunately, the same methodology is not valid for the quantization of the neuron's weights. While the neuron's weights may vary during the adaptation process, their variations and statistical dependencies are extremely case conditional and can only be obtained by solving (or almost solving) the adaptation problem first.

In order to overcome this lack of information and to provide a suitable quantized model for the neuron, this paper presents an approach that allows the evaluation of a codebook for the neuron's weight by assuming a few constraints in its input and output signals. More precisely, the input and output signals are considered enclosed in bounded sets. This assumption is intrinsically true for the neuron's output. The sigmoid function of the neuron provides the output saturation effect that allows its activation over specific operation regions of the input space. In addition, real signals always operate in finite dynamic ranges. This makes the bounded input consideration realistic.

We begin the weight quantization process by considering the simplest version of a neuron, a linear combiner, a bias weight and a smooth sigmoid function.

The neuron equation is:

$$\begin{aligned} v &= w^T x + b \qquad (2) \\ y &= f(v) \\ y, v, b \in \mathbf{R},\ & w, x \in \mathbf{R}^n,\ \ \|x\| \leq X_{MAX} \end{aligned}$$

x is the neuron's input whose maximum norm is X_{MAX}. The weight vector is w, the neuron's output is y, and b is the bias. f is a monotone increasing bounded real function (sigmoid).

In this analysis, the neuron is considered to be unbiased ($b = 0$). The quantization of the bias term will be discussed later in this section.

Letting $b = 0$, the first term of equation 2 can be rewritten as

$$v = \|w\| \, \|x\| \cos\phi, \tag{3}$$

where ϕ is the angle between the input vector x and the weight vector w.

Now, let $\alpha = \|x\| \cos\phi$ and equation 3 becomes

$$v = \|w\| \, \alpha, \quad |\alpha| \leq X_{MAX} \tag{4}$$

We assume that all possible projections of vector x onto vector w are equally probable. Therefore, the change of variables made in equation 4 is allowed.

In order to quantize w, we define a random version for equation 4:

$$V = WA \tag{5}$$

V, W and A are random variables with probability functions to be defined. They replace the deterministic variables v, $\|w\|$ and α in equation 4.

α has its dynamic range defined in equation 4. Since there is no further information concerning the statistics of α, all the possible positive values in its dynamic range will be considered equally probable. Thus, the random variable A takes the place of α in equation 5. Its uniform probability density function (pdf) is

$$A \sim U(0, X_{MAX}] \tag{6}$$

Negative values of α, including the origin, are not considered here to keep the calculations with random variables simple. The same procedure may be adopted considering its entire dynamic range. However, it will not change the final result.

For a given $A = \alpha$, the random version of w, i.e. W, should be defined in such a way that all possible values of v within its dynamic range can be evenly reproduced. Because the sigmoid function is bounded, there is a maximum for the absolute value of v so that no significant change in y occurs. Thus, the dynamic range for v can be bounded by a positive constant V_{MAX} and the random variable V is defined indirectly by the conditional event

$$V \mid A \sim U\,[0, V_{MAX}]\,, \tag{7}$$

which associated with equation 6 implies

$$V \sim U\,[0, V_{MAX}] \tag{8}$$

V and A and are independent and W can be expressed as

$$W = \frac{V}{A} \tag{9}$$

(note that A is always greater than zero). Hence, the random variable W will have its pdf given by:

$$g_W(w) = \begin{cases} \dfrac{1}{2}\dfrac{X_{MAX}}{V_{MAX}} & 0 \leq w \leq \dfrac{V_{MAX}}{X_{MAX}} \\ \dfrac{1}{2}\dfrac{V_{MAX}}{X_{MAX}}\dfrac{1}{w^2} & w > \dfrac{V_{MAX}}{X_{MAX}} \end{cases} \tag{10}$$

According to equation 10, values of w with norm smaller than $\frac{V_{MAX}}{X_{MAX}}$ are more likely to happen. Thus, in this region, the weight vector w has to be finely quantized. For values of w with norm greater than $\frac{V_{MAX}}{X_{MAX}}$ the discretization can be rough. In this case, the bigger the norm of w, the rougher the quantization can be.

In order to obtain an N-point vector-quantized codebook for w, a distribution of points in $\mathbf{R}^n$ for w that obeys equation 10 has to be generated. This can be performed in 3 (three) basic steps:

(1) Generate a set of unit vectors uniformly distributed on a unit sphere in $\mathbf{R}^n$. This can be done by first generating a uniform distribution of points in a unit hyper-cube in $\mathbf{R}^n$. Then, all the points with norm greater than 1 (one) are discarded and the remaining points are scaled to unit vectors. This will result in M unit vectors ($M \gg N$) evenly distributed on a sphere.
(2) Produce a set of M scalar points with distribution given by equation 10.
(3) The distribution of points for w is obtained choosing at random, pairs (scalar, unit vector) and scaling the corresponding unit vector by the value of the selected scalar. A vector or a scalar can be chosen once only. This will result in a distribution in $\mathbf{R}^n$ for w that obeys equation 10.

With the distribution of points produced, nearest neighbor encoding algorithms can be used to obtain an N-point vector quantizer of dimension n. Thus, vector-quantized weights for the neuron are obtained.

There is no specific rule for quantization of the bias b. Basically, it is bounded in a set (let us say $b \in [-V_{MAX}, V_{MAX}]$) and then, uniformly quantized. Alternatively, it could be quantized in such a way that its points, when mapped to the neuron's output y through the sigmoid function, with $w^T x = 0$, generate a uniform grid for y.

3. THE DESIGN OF NEUROINTERFACES USING ADAPTIVE NONLINEAR INVERSE MODELING

The concept of neurointerface was first developed by Widrow, *et al.* (1998) and may be thought of as an approximation of the system inverse model. Although such a statement may not be obvious,

in fact, an operator develops with his experience a set of causal rules that map standard behaviors into control actions (cognitive model), and this effect can be achieved by inverse modeling. Thus, a neurointerface tries to reproduce the actions of an experienced operator by using the system inverse model.

The nonlinear inverse modeling approach has been used for many years having evolved from linear inverse modeling (Widrow and Walach, 1996). Basically, the objective is to cancel the plant nonlinear dynamical effects by using a nonlinear device that can reproduce an approximate inverse of the plant. The term "approximate" is employed to emphasize that, in general, a nonlinear system does not possess an inverse. However, despite some pathological cases that might eventually exist, the methods of adaptive inverse modeling can often be applied to obtain acceptable inverse approximations of nonlinear systems.

The specification and design of a neurointerface use the fact that a nonlinear plant can be approximated by a neural network model, here represented by the function $f : \mathbf{R}^{p+q+2} \rightarrow \mathbf{R}$, of the form

$$y_{k+1} = f(y_k, \ldots, y_{k-p}, u_k, \ldots, u_{k-q}, W_M) \quad (11)$$
$$y_{k+1}, \ldots, y_{k-p}, u_k, \ldots, u_{k-q} \in \mathbf{R}, \quad W_M \in \mathbf{R}^{\ell_M}$$

Variables $y_k, \ldots, y_{k-p}, u_k, \ldots, u_{k-q}$ are the plant model inputs. y_{k+1} is the plant model output and W_M represents the vector quantized weights of all the neurons of the plant model.

The neurointerface can be regarded as a neural network approximation of the plant inverse model. a simple means for adapting a filter to be the inverse of a linear plant is shown in figure 1.

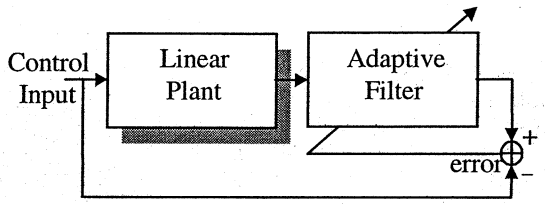

Fig. 1. Inverse modeling of a linear plant.

The first step in a nonlinear neurointerface design is to obtain a neural network model for the plant as defined in equation 11 and then, use it to obtain a neural approximation for the plant inverse (neurointerface). The neural model can be trained with a set of input-output data either acquired from the real plant or obtained from the plant mathematical model (if available).

The final step, shown in figure 2, is to train the neurointerface to compute an approximate inverse for the obtained plant neural model. Like the

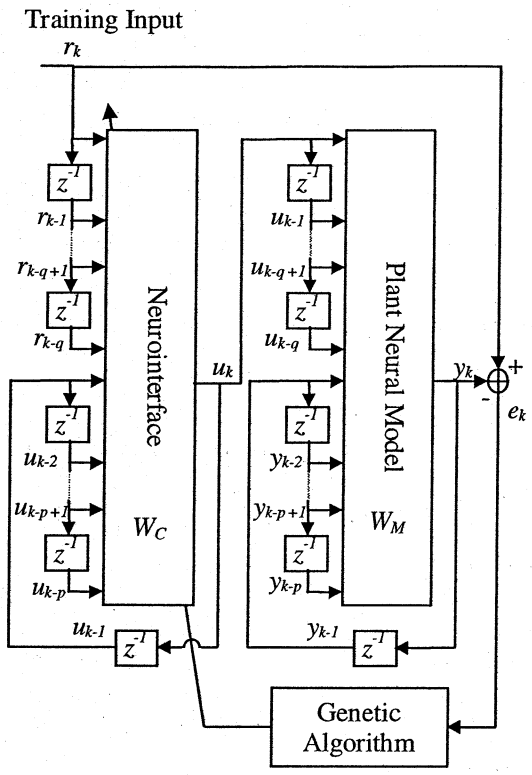

Fig. 2. Training a neurointerface.

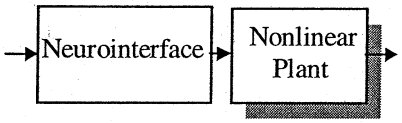

Fig. 3. A cascade of the trained neurointerface and the plant.

neural model case, the neurointerface computes a function $g : \mathbf{R}^{P+Q+1} \rightarrow \mathbf{R}$, of the form

$$u_k = g(u_{k-1}, \ldots, u_{k-Q}, r_k, \ldots, r_{k-P}, W_C) \quad (12)$$
$$u_k, \ldots, u_{k-Q}, r_k, \ldots, r_{k-P} \in \mathbf{R}, \quad W_C \in \mathbf{R}^{\ell_C}$$

Variables $u_{k-1}, \ldots, u_{k-Q}, r_k, \ldots, r_{k-P}$ are the neurointerface inputs. u_k is the neurointerface output and W_C represents the vector quantized weights of all the neurons of the neurointerface.

The neurointerface shown in figure 2 is a feedback system, a recurrent neural network, that is trained off-line. The training input signal is r_k, which could be a random noise with suitable choice of spectrum. The plant neural model is nonlinear, and the neurointerface is trained to be its inverse. The cascade of the two should be linear. A cascade of the neurointerface driving the actual plant is shown in figure 3. The weights of the neurointerface are represented by elements of the VQ codebook, and are trained by a genetic algorithm.

An adaptive linear control scheme is shown in figure 4. It is based on an adaptive inverse control technique described by Widrow and

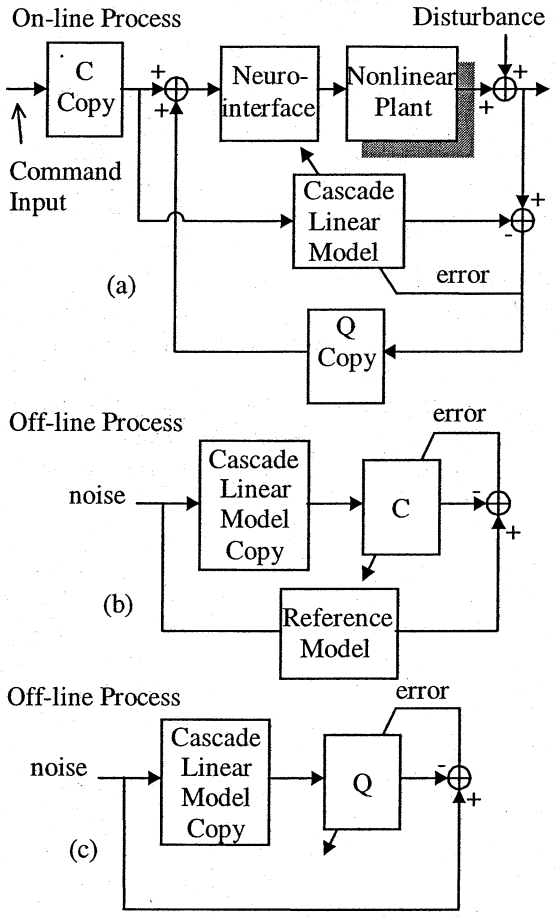

Fig. 4. An adaptive inverse control system with a neurointerface: (a) the entire system; (b) off-line controller adaptation; (c) off-line disturbance canceller adaptation.

Walach (1996). In this scheme, an equivalent linear model for the cascade of the neurointerface and the nonlinear plant is identified in real time. Then, using a digital copy of this cascade linear model, the controller C and the disturbance canceler Q are calculated off-line. The off-line processes can run much faster than real time, so that as the cascade linear model is identified, Q and C are immediately obtained.

The neurointerface is able to cancel most of the nonlinear effects the plant may have and indeed, it can be used in combination with adaptive linear control schemes for the control of nonlinear plants.

4. EXPERIMENTAL RESULTS: TRAINING A NEUROINTERFACE WITH GA

Backing a truck and trailer along a path is a difficult task for all but the most skilled truck drivers. This section briefly presents the experimental results of a neurointerface, trained with GA, that reduces the trailer truck operation exercise to a much less complex problem. A scaled model truck

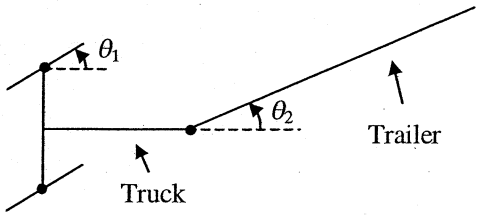

Fig. 5. The steering angle, θ_1, and the angle between truck and trailer, θ_2.

connected in a single-trailer configuration (see last page picture) is utilized as a case study. The length of the truck is 0.4 m and the length of the trailer is 0.78 m.

The neurointerface may be considered as a black box that takes commands from the driver (desired direction of the trailer back part) and provides the necessary actions (steer the front wheels) in order to achieve such a goal. The desired direction of the trailer back part is related to the angle between truck and trailer, θ_2, and the front wheel steering angle, θ_1 (see figure 5).

The neurointerface has, as its inputs, the truck speed, the desired value of θ_2, and the previous value of the neurointerface's output, θ_1.

The neurointerface was designed following the steps described in section 3. Acquired data from the scaled model truck were used to obtain a neural mode that was used for the neurointerface training.

The neurointerface alone could have been used with human input to control the nonlinear plant in the manner illustrated in figure 3 if it were not for plant disturbance. To deal with dynamic control and plant disturbance simultaneously and independently, the system of figure 4 (a) was employed. In it, an adaptive filter was trained to model the cascade of the neurointerface and the nonlinear plant. Another adaptive filter, Q, was trained for disturbance cancelling, as diagrammed in figure 4 (c). For model reference control, another adaptive filter, C, is trained as shown in figure 4 (b) and it is utilized as in figure 4 (a) to provide an overall dynamic response in accord with a designer-chosen reference model shown in figure 4 (b). This system was successfully used for steering the physical scaled model truck and trailer going backwards.

Good results were obtained training the neural model with 3 layers: 7 neurons in the first layer, 7 neurons in the second and 1 linear neuron in the last layer. The neurons were vector quantized following the steps described in section 2. Each neuron has its weight vector quantized to 10 bits (codebook with 1024 values). The bias was uniformly quantized using a 5-bit scalar quantizer. The neurons were quantized using 15-bit words.

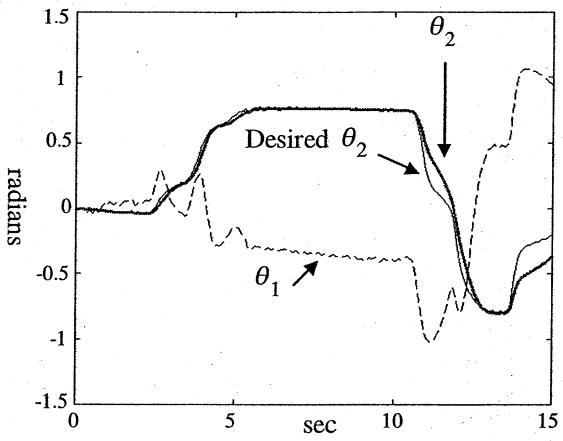

Fig. 6. Experimental results: desired behavior of θ_2, real behavior of θ_2, and neurointerface's steering output, θ_1.

To run the GA, simple reproduction, crossover and mutation operations were employed, see (Kosa, 1992). A fixed population size of 50 individuals was used and the maximum number of generations was 200. The same network configuration used for the neural model had been used in the neurointerface.

The experimental results are shown in figure 6. They correspond to sequences of data acquired from the model truck moving backwards on an average speed of 0.5 m/s. The direction over time of the truck and trailer came very close to the desired direction.

5. CONCLUSIONS

This article presents a new approach for the quantization of the weights of a neuron using Vector Quantization. It shows how Genetic Algorithms (GA) and Vector Quantization (VQ) can be applied to the training of neurointerfaces. The GA in combination with the proposed vector-quantized neuron are used to train a neurointerface that helps a human operator back up a scale model truck with a single trailer.

This is an introductory work and a great effort will be needed to improve the utilization of vector-quantized neurons in combination with GA for the training of neurointerfaces. However, the general aspects covered in this paper combined with the excellent quality of the experimental results lead one to conclude that the full utilization of this approach for neural network training will be very useful in the future.

6. REFERENCES

David, J. M. and L. Davis (1989). Training feed-forward neural neworks using genetic algorithms. In: *Proceedings of the International Joint Conference on Artificial Intelligence* (Morgan Kaufmann, Ed.).

Gersho, A. and R. Gray (1992). *Vector Quantization and Signal Compression.* Kluwer Academic Publishers.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems.* Ann Arbor, MI: University of Michigan Press.

Kosa, J. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* A Bradford Book, MIT Press. Cambridge, MA.

Rumelhart, D. E., G. E. Hinton and R. J. Williams (1986). *Learning Internal Representations by Error Propagation. Parallel Distributed Processing.* Chap. 8. Vol. 1. MIT Press. Cambridge, MA.

Widrow, B. and E. Walach (1996). *Adaptive Inverse Control.* Prentice Hall PTR. Upper Saddle River, NJ.

Widrow, B., E. P. Ferreira and M. M. Lamego (1998). Neurointerfaces for human-machine real time interaction. In: *Proceedings of the IFAC Workshop for Real Time Algorithms.* Cancun, Mexico. pp. 131–136.