Randomly Linearized Non-linear Neural Networks (for Determining Articulatory Positions From Speech)

*Olivier Rioul**
Bishnu S. Atal

AT&T Bell Laboratories
Murray Hill, New Jersey 07974

ABSTRACT

Using techniques related to a two-layer Perceptron Neural Network representation, investigations of the relationship between acoustic parameters and positions of different articulators in the vocal tract were made on a computer-simulated Mermelstein articulatory model. After presenting different neural net representations such as random waveform mode decomposition connected to a continuous Kanerva model, and various non-linear regression techniques, we introduce the Randomly Connected Neural Networks representation and the associated "*Randomly Linearized Adaptation*" Algorithm. It was found that, using this technique, the determination of articulatory positions can be achieved with good accuracy, using different acoustic parameters such as linear predictor coefficients, spectrum, cepstrum, reflexion coefficients and line spectral frequencies.

Keywords:

Two-Layers Perceptron, Back-propagation, Kanerva Model, Least Mean Square Algorithm, Acoustic Parameters, Mermelstein Articulatory Model, Articulatory Parameters.

---

* IGE - Ecole Nationale Supérieure des Télécommunications - 46 rue Barrault - 75013 PARIS, France.

1. Introduction.

The current enthusiasm for artificial neural networks finds its motivation in the hope that investigations in neurobiology may some day reveal the underlying principles related to the human brain. They would allow us to build modelized (and until now quite simple) systems that imitate the characteristics of the powerful tasks performed by human beings, for solving problems where sequential algorithms lack some capabilities: pattern recognition, or speech understanding is one example.

One of the numerous explanations given for this motivation is that neural net systems seem to achieve unusual performances: while given uncomplete information (*"training set"*), they perform a noise-resistant task in a parallel and high speed way.

The neural network design is of practical interest to us because it gives a flexible, new representation of an underlying non-linear mapping; it can be used to perform adaptive non-linear signal processing. Not surprisingly, many non-linear systems have been analysed using predominantly linear analysis, which is not likely to perform as well as an appropriate non-linear analysis.

The two-layer perceptron representation, described below, provides a natural generalization of linear processing as a Taylor-like expansion does. Our model is continuous, that is, inputs and outputs are not restricted to binary values: they are instead linear elements that can take all real values from a normalized range $[-1,1]$. They provide thus a dynamic range more closely adapted to the real values of the mappings we considered for applications. The non-linear transfer functions (sometimes referred to as *"hidden nodes"*) use a continuous, bounded, non-linear (and non-polynomial) basis function. This provides a mode decomposition of the multidimensional mapping considered, into basis functions labeled by amplitudes and shifts coefficients. These coefficients are also referred as "1^{st} *layer connections weights*". The preparation of the data and learning principles we use are explained in this paper.

The originality of the algorithm we use is that unlike most time series' expansions, the coefficients used to describe the collection of the basis functions are not preset to a particular fixed structure. Unlike, for example, in the back-propagation algorithm[1], these coefficients are not adjusted by adaptation to the data, which demands many iterations and a good accuracy. Instead, the different shapes of the basis functions are chosen at random. This has been proven to give an efficient mode decomposition as the basis functions are likely to scan the output space, and has allowed us to use a straight-forward, fast and linear adaptative algorithm to find a reasonable local minima of the cost function (commonly taken as the variance error of the actual output). This scheme is also closely related to a modified Kanerva Model for continuous mappings.

We have used and tested this algorithm on many different mappings, synthetic or real (see also [5]). In this paper we present the results obtained for determining articulatory parameters from speech. The mapping used is not a straight-forward one, and therefore is likely to give us more insight about real problems involving non-linear networks. Although a particular model (the "*Mermelstein*" model) was used, and acoustic parameters were taken from a model estimation of the transfer function of the vocal tract, things can be investigated further, without any particular difficulty (except as for the preparation of the data), using real speech.

2. Non-linear Decomposition of multidimensional mappings.

From now on we shall restrict ourselves to a particular functional representation of a non-linear system. Let's begin with a particular case. Assume a known multidimensional mapping $\underset{i=1,..,m}{(x_i)} \rightarrow \underset{j=1,...,n}{(y_j)}$ is given. Typically x_i and y_j are also functions of time: $x_i=x_i(t)$ and $y_j=y_j(t)$. Moreover, a given $y_j(t)$ may depend on the history up to t of the input variables $x_i^t(\tau)=x_i(t-\tau)$. In our case, $\underline{x}$ is any kind of m-dimensional acoustic parameter vector, and $\underline{y}$ is a model-defined set of articulator positions in the vocal tract. The mapping involved is complicated but is deterministically calculable from an appropriate model. Then the question that arises is how to invert this mapping. For the sake of simplicity and efficiency, several assumptions are to be made. First, we restrict ourselves to time-invariant mappings:

$$\underline{y}(t) = \mathbf{M}(\underline{x}(t)) \tag{1}$$

Second, we consider only mappings without memory. Time will not play a role in our discussions, despite the possibly many advantages of time adaptative signal processing here. Third, we assume that the inverse of **M** is well defined in the ranges of the input and output values considered. This asumption, although impossible to verify for complicated mappings, can be made less constrained: The representation used can determine by itself the quasi-invertibility structure ($\mathbf{M}^+$), provided that $\mathbf{M} \,.\, \mathbf{M}^+$ is close to identity. After appropriate normalization, we can assume that the values of all the outputs and inputs lie between -1 and $+1$. The complexity of the mapping is consequently reduced to a $n \times m$ dimensional non-linear mapping **M**.

In order to invert this mapping, we need a good non-linear representation of $\mathbf{M}^{-1}$. We didn't consider the possible *apriori* assumptions on the correlation between the articulatory parameters, so that for our classes of representations of $\mathbf{M}^{-1}$ the different outputs will be determined independently. Thus we can represent one projection (for a given output O_j) of the inverse system by a multi-input mapping:

$$\mathbf{M}_j^{-1}: \underset{i=1,..,n}{(I_i)} \in [-1,+1]^n \rightarrow O_j \in [-1,+1] \tag{2}$$

An *estimated* output $\hat{O}_j$ is an unknown non-linear mapping of all the inputs $\underline{I}$, that can be represented in several ways. We will describe a few of them. Once a precise representation with an appropriate set of unknown parameters is given, these unknowns can be determined by minimizing a given *cost function* over this set of parameters. A cost function in the form:

$$\mathbf{E} = \sum_j \mid O_j - \hat{O}_j \mid^2 \qquad (3)$$
$$where \quad \hat{O}_j = \mathbf{M}_j^{-1}(I_1, \cdots, I_n)$$

will provide an approximation in the $\mathbf{L}^2$ space of the outputs with relative error (inverse of signal over noise ratio) equal to $\mathbf{E} \,/\, \mid \underline{O} \mid^2$.

An additional restriction is imposed by the implementation: Only a discrete subset of all the possible inputs and outputs can be implemented, via indirect knowledge of **M**, which provides incomplete information for a particular determination. Therefore the input and output spaces to be considered (for a particular determination of $\mathbf{M}^{-1}$) is finite and discrete, with associated $\mathbf{L}^2$-norm defined as

$$\mid \underline{I} \mid \;=\; \sqrt{\sum_{i=1}^{n} \sum_{\tau=1}^{N} \mid I_i^{\tau} \mid^2} \qquad (4)$$

where the index i refers to the dimension and τ to the discrete index covering the space. In some of our implementations, the dimension of both the input and output space was 10, and N took values between 5,000 and 10,000. In the neural network's terminology, the couple (Input Space, Output Space) is known as the *training set.* Several methods are possible to minimize **E** in the most general case. A common approach consists of a gradient descent search: starting from a random set of parameters, **E** decreases as parameters a_i are changed according to:

$$a_j \;=\; -\varepsilon \frac{\partial \mathbf{E}}{\partial a_j} \qquad (5)$$

where ε is small. In general, this provides only a local minimum for **E**. Whether this local minimum is close to the global minimum is unpredictible.

The increasing effort to implement techniques of analysis for non-linear systems is predominantly based on two different types of representation designs of these systems: polynomial systems and the "neural network" approach. We will therefore briefly present them both, and show some connections between the two.

2.1 Linear Case.

The predicted value is a linear combination of the different inputs:

$$\hat{\underline{O}} = \underline{\mathbf{A}} \cdot \underline{I} + \underline{a} \tag{6}$$

where $\underline{\mathbf{A}}$ is a $m \times n$ matrix, $\underline{I}$ the n-dimensional input vector, and $\underline{a}$ a m-dimensional vector. As shown below, this representation can be seen as a linear one-layer perceptron, with no "hidden units". The constant part $\underline{a}$ (the vector of "thresholds" in the language of neural nets) can be integrated in the matrix $\underline{\mathbf{A}}$ if the inputs are added to a constant input I_0 always fed by the constant 1. Determining the parameters $\underline{\mathbf{A}}$ and $\underline{a}$ is straight-forward.

2.2 Polynomial Series.

Polynomial series are a straight-forward extension of the linear estimation, where polynomial non-linearities are added in the expansion. O_j is estimated by series of inputs in the form:

$$\hat{O}_j = a_j + \sum_{i=1}^{n} a_{ij} I_i + \sum_{i_1,i_2} a_{i_1,i_2,j} I_{i_1} I_{i_2} + \cdots \tag{7}$$
$$\text{for } j = 1, \ldots, m.$$

The number of parameters for a polynomial expansion of order k is:

$$\frac{(m+k)(m+k-1)\ldots(m+1)}{k!}$$

and grows exponentially with the dimension m of the output. In this case, the output is estimated by a linear combination of basis functions (the polynomial non-linearities). In order to determine the unknown parameters of the representaion minimizing the cost function **E**, a simple least mean square algorithm can be used: In fact, for any expansion of the form:

$$\hat{O}_j = \sum_{i=1}^{N} a_{ij}\, b_i(I_1, \ldots, I_n) \tag{8}$$

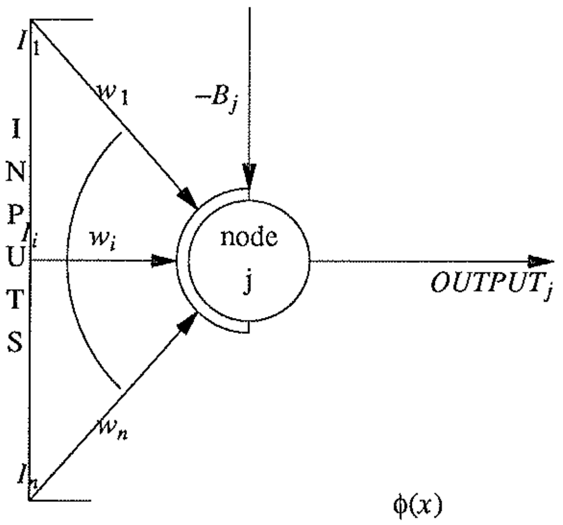
where b_i are the basis functions, minimizing **E** (sum of the quadratic errors over the training set $\underset{\tau}{(\underline{I}^{\tau}, O_j^{\tau})}$) over the parameters a_{ij}, is achieved by letting the derivatives of **E** with respect to a_{ij} equal 0. This provides a set of linear equations, and the coefficient matrix is the Gramm matrix of the basis functions:

$$G_{ij} = \sum_{\tau} b_i(I_1^{\tau}, \ldots, I_n^{\tau})\, b_j(I_1^{\tau}, \ldots, I_n^{\tau}) \tag{9}$$

Then $G^{-1} \cdot \underline{b}^t . \underline{O}$ gives the set of parameters minimizing **E** exactly (in this case a global minimum is found).

2.3 Principles of Neural Networks.

The neural network structure consists of simple nodes (or "neurons") connected through links. Each node can be seen as a basic non-linear computational element. It may have several inputs, and its output (or state) depends on the states of many other nodes. Its task is simply to pass the result of a linear combination of its inputs through a non-linear one-dimensional transfer function ϕ. The weights of the linear combination are associated with the weights of the connections feeding the node. The biases are associated with each node, and modify the action of the non-linearity by shifting its characteristic. Because they are often part of the representation parameters, one can see biases as special connection weights fed by an "always-1" input (see figure **1**).



$$OUTPUT_j = \phi(\sum_{i=1}^{n} w_{ij} I_i - B_j)$$
$$\Phi(\underline{x}) = (\phi(x_j))_{j=1..n}$$

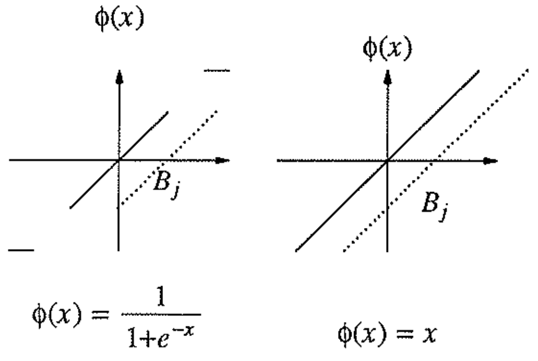


Figure 1. The non-linear neuron principle.

The ϕ function (which is the same for all non-linear nodes) may actually be chosen in an arbitrary way -continuous or smoothed, invertible or non-invertible. Most widely used is the "sigmoid" form as in $\phi(x) = 1/(1+e^{-x})$. For small inputs and non-active connections, a sigmoid presents a linear characteristic. For large inputs, the saturation part limits the range of the output, unlike a polynomial non-linearity. Our implementation allows the choice of several kinds of such ϕ's. It appeared to us that the behavior of the systems involved doesn't depend on the particular form of the (bounded) non-linearity, provided that ϕ doesn't oscillate too much, or remain flat.

It would be tedious and practically almost impossible to handle a network in which all possible connections are allowed. The most interesting behaviors have been found in well-defined, restricted neural networks. A common restriction is to consider only layered networks, that is networks with structures presenting successive layers of nodes, with most connections lying between two consecutive layers. Inputs and outputs are generally present in the first (bottom-most) and last (top-most) layers, respectively. Moreover, we will assume that there is no intra-layer connectivity and that the inter layer "all-to-all" connectivity is feedforward, i.e. each node in a given layer feeds its output to all nodes in the next layer, and does not feed back, nor feeds to its neighbors.

These assumptions are somewhat restrictive but allow simple and efficient learning procedures to be achieved. The resulting structure is often referred to as "*multi-layer perceptron*". One can see n-layer perceptrons as special "sandwich" representations of non-linear mappings in the matrix form (where input and output nodes are linear):

$$\underline{\hat{O}} = \underline{\mathbf{A}}_n \,.\, \Phi \,.\, \underline{\mathbf{A}}_{n-1} \,.\, \Phi \,.\, \cdots \,.\, \underline{\mathbf{A}}_2 \,.\, \Phi \,.\, \underline{\mathbf{A}}_1 \,.\, \underline{I} \tag{10}$$

where Φ is a non-linear multi-dimensional operator: $\underline{\Phi}(\underline{x}) = (\phi(x_i))_{i=1..n}$

Perceptrons have been proven to be efficient to perform associations (where input and output patterns may be completely arbitrarily independent: *hetero-association*). These associations input$\rightarrow$output are taught to the system during the *training phase*, in which the different parameters (connection weights) are adjusted. Most of the training procedures have in common the fact that at each successive instant, the system is presented with one association pair taken from the training set consisting of one input and one output vectors: then an interative *training algorithm* is performed to modify the connection weights (see figure **2**).

Note that many independent input-output pairs can be presented and memorized simultaneously, but recalled independently while testing or using the final system. Some algorithms can perform differently, depending on the order of the association pairs present in the training set. The type of training algorithm is essential: it determines the (good or bad) ability to interpolate the associations it has been given during training. Several training algorithms relevant for our applications are presented later in this paper.

Once the system is trained, it can be tested in different ways. For our purposes, the testing phase must be able to determine the ability of the present trained network to generalize (or interpolate) the association pairs present in the training set, in order to perform a comparable task on different data.

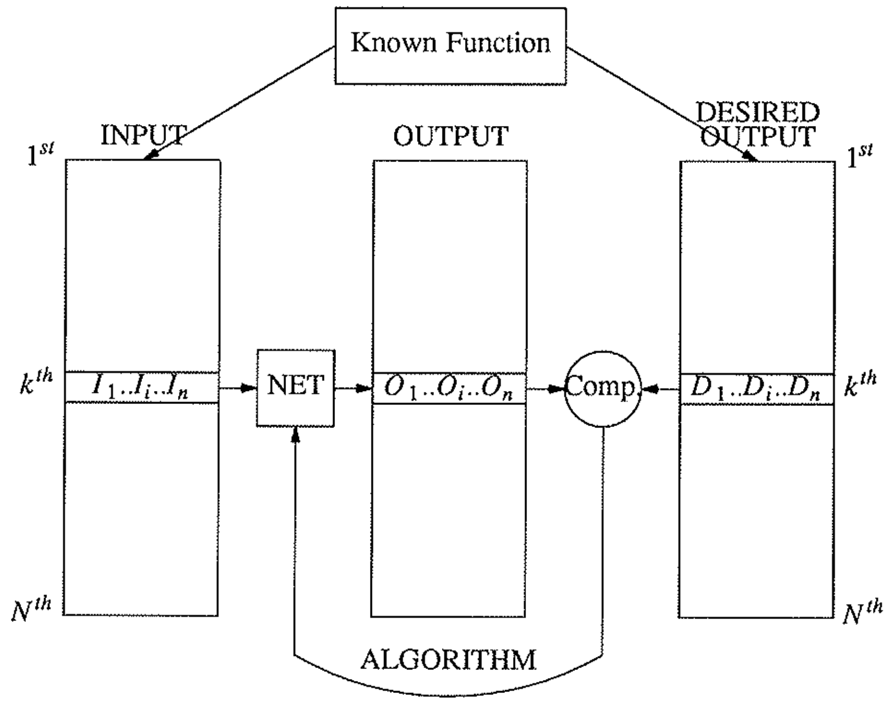


Figure 2. The Training Procedure

Therefore, we tested the data by producing another *test set* containing different input-outputs pairs, feeding the network with these inputs to collect the estimated outputs, and comparing the results with the real outputs of the test set. The quadratic error function:

$$\mathrm{E} = \sum_j | O_j - \hat{O}_j |^2 \qquad (11)$$
$$\textit{where} \quad \hat{O}_j = \mathbf{M}_j^{-1}(I_1, \cdots, I_n)$$

gives a reasonable estimation of the error rate in the output space. Some kind of error determination can also be performed on the input space by retrieving the input values back from the estimated outputs (if one disposes of the exact inverse mapping) and comparing them to the desired inputs of the test set.

2.3.1 Linear Estimation and polynomial series.

The structure of perceptron that is amenable to a linear estimation procedure (§**2.1**) is simply a single-layer perceptron with all nodes being linear. This, of course, is a degenerate case, but shows that one can always add a linear and constant term in the representation of estimated outputs in a more complicated structure, by adding feedforward connections from the linear input nodes to the linear output nodes. The constant part is, as mentionned before, represented as always-1 inputs (see figure **3.0**).

With the linear case, the ability to separate one output pattern from the other memorized ones is low compared to when non-linear nodes are used, because the output is a linear combination of all input training values. Therefore, once the network is trained, inputing the training data will never provide a good approximation of the output training patterns (the estimation of the multidimensional mapping is made by an hyperplane), except if the mapping considered happens to be very close to linear.

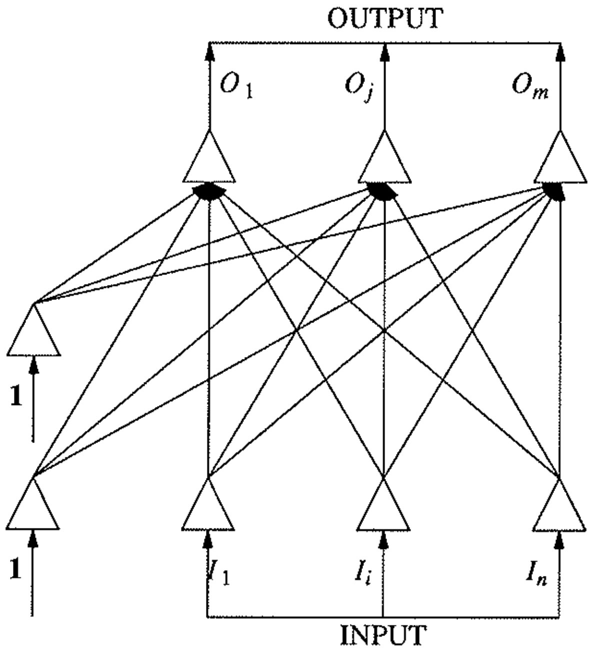


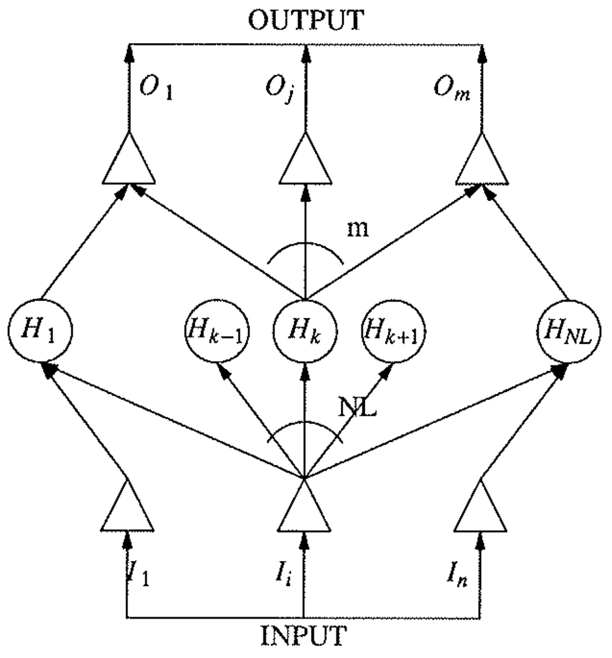
Figure 3.0. Some like it as: Single Layer-Perceptron.

Using a two-layer perceptron with linear nodes wouldn't work very well either: The outputs of many hidden nodes (which would be linear combinations of the inputs) would be linearly dependant, thus providing an unstable learning algorithm.

Estimation with polynomial expansions can also be seen as a two-layer perceptron, where the connection weights of the bottom layer are fixed and adjusted so that the output of the intermediate node j is the j^{th} polynomial image of the input pattern. Note that in this case different types of non-linearities are used, which can be reduced to x, x^2, x^3, etc. The introduction of many different non-linearities improve the discrimation power of the representation. Each intermediate node is representative of an independant local representation of the mapping.

2.4 The Back-Propagation algorithm.

In our implementations of the Back-Propagation Algorithm a two-layer perceptron representation was used. We have added a constant linear part by superposing the structure shown in **Fig. 3.1, 3.2** with a linear single layer perceptron from the inputs to the inputs.



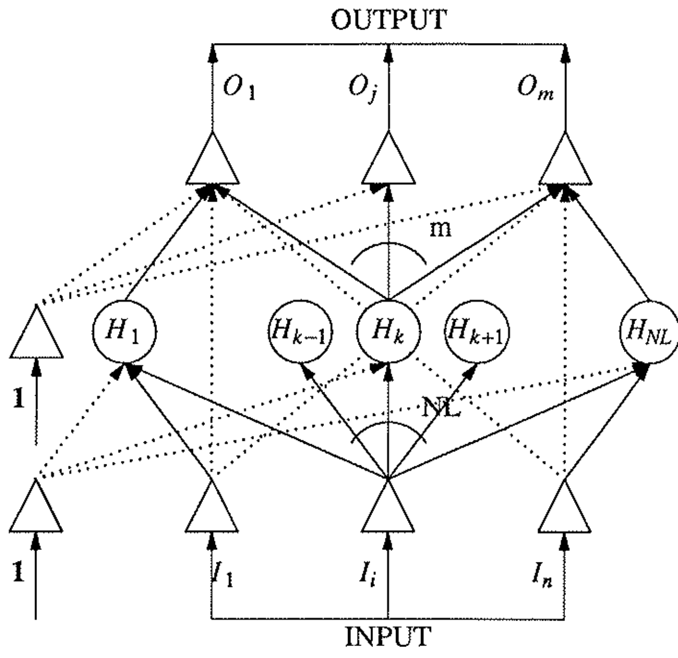


Figure 3.1. A two Layer-Perceptron.

Figure 3.2. Final neural net structure with constant and linear part.

Our representation then becomes:

$$\underline{\hat{O}} = \underline{\mathbf{A}} \cdot \underline{\Phi} \cdot \underline{\mathbf{R}} \cdot \underline{I} \tag{12}$$

without any constant and linear part, and:

$$\underline{\hat{O}} = \underline{\mathbf{A}} \cdot \underline{\Phi} \cdot (\underline{\mathbf{R}} \cdot \underline{I} + \underline{\mathbf{r}}) + \underline{\mathbf{L}} \cdot \underline{I} + \underline{\mathbf{c}} \tag{13}$$

with constant and linear part. Φ is still defined as:

$$\underline{\Phi}(\underline{x}) = (\phi(x_i))_{i=1..n}$$

and $\underline{\mathbf{A}}$ and $\underline{\mathbf{R}}$ are the matrices associated to the connection weights of the bottom and top layer, respectively. We can integrate in the mathematical formalism the vectors $\underline{r}$ and $\underline{c}$ by introducing two always-1 input nodes as shown in figure **3.2**: An estimated output is then represented as a linear combination of the inputs and the outputs of the *NL* intermediate nodes:

$$\hat{O}_j = \sum_{k=1}^{NL} \mathbf{A}_{kj} H_k + \sum_{i=1}^{n} \mathbf{L}_{ij} I_i \tag{14}$$

$$i = 1,2, \cdots m.$$

$$\textit{where} \;\; \mathbf{A}_{oj} = \mathbf{c}_j \,, \;\; H_o = 1.$$

and the hidden nodes' outputs H_k are calculated from:

$$H_k = \phi(\sum_{l=o}^{n} \mathbf{R}_{lk} I_l) \tag{15}$$
$$\text{for } k = 1,2, \cdots NL$$
$$\textit{where } \mathbf{R}_{ok} = \mathbf{r}_k \,, \; I_o = 1.$$

The Back-Propagation Algorithm can be thought as a particular minimization of the quadratic error function $\mathbf{E} = |\underline{\hat{O}} - \underline{O}|^2$ (see equation (5)): Computing the gradient of $\mathbf{E}$ with respect to the A_{kj} leads to:

$$\frac{\partial \mathbf{E}}{\partial A_{kj}} = 2(\hat{O}_j - O_j)\frac{\partial \hat{O}_j}{\partial A_{kj}} = 2(\hat{O}_j - O_j)H_k \tag{16}$$

The term "back-propagation" comes from an interpretation of the derivative of $\mathbf{E}$ with respect to the first layer connection weights R_{lk} (we use the notation ϕ' for the derivative of ϕ):

$$\begin{aligned} \frac{\partial \mathbf{E}}{\partial R_{lk}} &= 2\sum_j (\hat{O}_j - O_j)\frac{\partial \hat{O}_j}{\partial H_k}\frac{\partial H_k}{\partial R_{lk}} \\ &= 2\sum_j (\hat{O}_j - O_j) A_{kj} \frac{\partial H_k}{\partial R_{lk}} \\ &= 2[\sum_j (\hat{O}_j - O_j) A_{kj}]\; \phi'[\phi^{-1}(H_k)]\; I_l \end{aligned} \tag{17}$$

For each iteration of the algorithm, the weights will be changed according to (5):

$$\Delta A_{kj} = -\varepsilon \frac{\partial \mathbf{E}}{\partial A_{kj}} \quad \text{and} \quad \Delta R_{lk} = -\varepsilon \frac{\partial \mathbf{E}}{\partial R_{lk}}$$

Depending on the gradient descent algorithm used, ε can be fixed for both layers, or can be adjusted at each step such that the gradient descent is maximal. If we let δ_j be the ratio of the weight modification from point i to point j (that is either ΔA_{ij} or ΔR_{ij}) and the output in i (that is either I_i or H_i), then the δ's in the top layer are proportional to the linear error of the estimated and desired output: $(\hat{O}_j - O_j)$ and the δ's in the bottom layer will be: $(\sum_j A_{kj}\delta_j^{toplayer})\; \phi'(\phi^{-1}(H_k))$, that is the *back-propagated* errors δ_j of the top layer (as if the arrows were turned back) multiplied by a correction term due to the non-linearity used. If this non-linearity happens to be:

$$\phi(x) = \frac{1}{1+e^{-x}}$$

then this correction term is reduced to $H_k(1-H_k)$, that can be seen as an anti-saturation term, 0 at the saturation of ϕ, $\frac{1}{4}$ where the characteristic of ϕ is close to linear. Therefore, first layer connection weights are not modified during training if the corresponding non-linear node remains saturated. The connection weights are usually initialized to very small random values before training, in order not to saturate the network at the begining of the training procedure.

The Back-Propagation algorithm can be seen as a non-linear regression where an ouput $\hat{O}_j$ is decomposed into a linear combination of many non-linear basis functions in the form: $\phi(\sum_{l=1}^{n} \mathbf{R}_{lk} I_l + \mathbf{r}_k)$. For this purpose, it can be thought of as some kind of curve fitting method, where non-linear sigmoids are added in the interval $[-1,+1]$ to form the function $\hat{O}_j(I_1, \ldots, I_n)$. In the decomposition of this function into sigmoids, the $\mathbf{R}_{lk}$ adjust the "scaling" or dilatation parameter for each coordinate l of the input, and the $\mathbf{r}_k$ adjust the shifts. This kind of decomposition can actually be achieved by all kinds of two-layer continuous perceptrons; it is not due to the back-propagation algorithm itself.

The originality of the back-propagation algorithm is that the coefficients of translation $\mathbf{r}_l$ and dilatation $\mathbf{R}_{lk}$ are actually adapted to the training data, together with the multiplicative coefficients of the sigmoids $\mathbf{A}_{kj}$. Because saturated sigmoids have a tendency not to be modified during training (see above), very sharp non-linearities will not be obtained easily. Therefore, a perceptron trained with back-propagation can only produce smoothed representations of mappings, in the same manner as functional approximation that maximizes smoothness (e.g. using splines).

This kind of mode decomposition, where basis functions are labeled by translation and scale parameters, is reminiscent of several Haar-like analyses. Consider $\psi(x)=\phi(x)-2\phi(x-1)+\phi(x-2)$. It is possible to choose the connections weights of the first layer such that the given mapping is decomposed into the family of basis functions in the form: $\psi(2^a x-b)$, where a and b can be any pair of integers. For a degenerated form of ϕ:

$$\phi(x) = \begin{cases} 0 & \text{for } x \textit{ negative} \\ 1 & \text{for } x \textit{ positive} \end{cases} \tag{18}$$

this system is called the Haar system and form a complete (orthonormal) basis of all signals of $[-1,1]$ of finite energy. In this case a is a scale (or "frequency") factor whereas b plays the role of a translation step in space. Two-layer perceptrons thus are able to represent a general mapping, as long as there are enough non-linear nodes and as long as the choice of sigmoids and of the learning algorithm is efficient enough.

2.5 A Continuous Karnerva Model.

A continuous approach of the Kanerva's Associative Memory Model has been studied in [2]. In this attempt, the norm chosen to the locations' matching procedure, instead of being the Hamming distance between two bit patterns, was adapted to continuous values: the L^2 norm with respect to the input space is one of the possible choices (although [2] prefers the L^{128}

norm). We choose the L^2 norm here because it allows a straight-forward extension to new ideas, as explained later. Independently of the norm used, the idea remains the same: it is much more efficient to train a parallel two-layer network without any adaptation to the data for the first layer (unlike the Back-Propagation algorithm). Of course, one needs to find a good non-parameteric distribution for the first layer connection weights' values.

In order to provide this, the Kamerva model is based of a location *matching procedure*:

1. First, a large number of random vectors of dimension of the input are created in the normalized range. Unlike [2], the random generator can simply provide random values for each coordinate. These values will remain constant during the learning algorithm.
2. Then, for each input pattern, the distances between this input pattern and all the fixed (random) vectors, say, with respect to the L^2 norm, are computed (see figure 4).

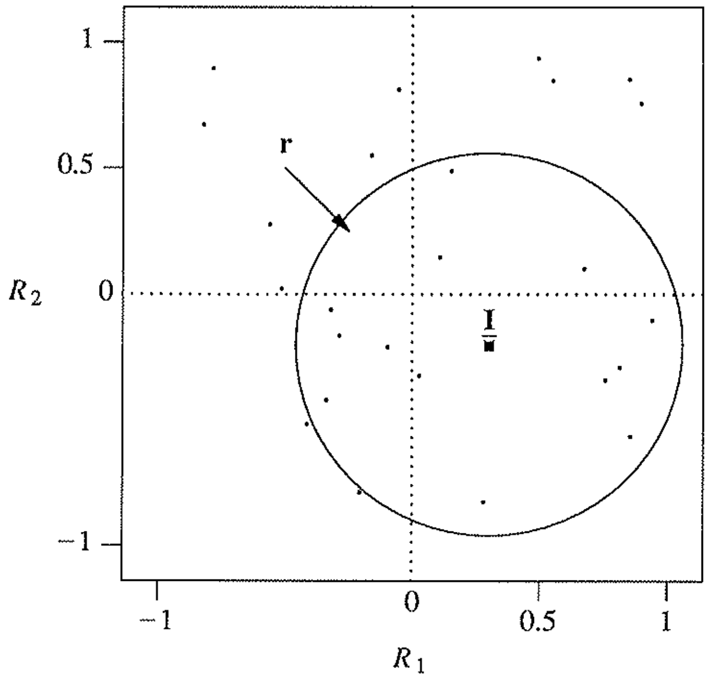


Figure 4. Location matching procedure (in two dimensions).

3. If a distance appears to be more than a certain threshold (called *the activation radius* **r**, fixed during the algorithm), then a corresponding intermediate node (a *location address*) is *activated* (output set to 1), otherwise this node is not activated (output set to 0). **r** is determined such that a certain (low) percentage of activation is observed on average.
4. The final output is then estimated as linear combinations of the activated nodes (or locations). For that purpose, any linear adaptation can be performed.

In other words, in order to estimate the activation of a given location, one has to simply compute:

$$H_k = \phi(\mathrm{r}^2 - |\underline{I}-\underline{R}_k|^2) \tag{19}$$

where ϕ is defined as a step function (or hard threshold sigmoid), as in equation (18). Any output can then be determined by a least mean square algorithm as linear combinations of these H_k. The notations used here are intentional: they show that H_k can actually be thought of as outputs of the intermediate layer's non-linear nodes, and that the R_k's are nothing but the first layer's connection weights. Indeed, on can write from equation (19):

$$H_k = \phi(2\ {}^t\underline{I}\ .\ \underline{R}_k\ -\ \mathrm{r}'^2) \tag{20}$$

where $r'=\sqrt{|\underline{I}|^2+|\underline{\mathbf{R}}_k|^2-\mathrm{r}^2}$ appears as a given threshold for the intermediate non-linear nodes, and where $\underline{\mathbf{R}}_k$ is seen as the k^{th} line of the matrix $\mathbf{R}$ in equation (12).

This kind of procedure involves merely some matrix computation and inversion for the linear adaption process. It therefore allows us to take many more intermediate nodes than possibly could be handled by the back-propagation algorithm. Because of the large number of intermediate nodes involved, the norms of $\mathbf{R}_k$ can be estimated as a constant number not depending on the training set or on k, for a given mapping. Thus the threshold $\mathbf{r}'$ is predominantly dependent on the norm of the input pattern. We will discuss similar properties while presenting the randomly linearized adaptation procedure. Note that, however, as is true also for other techniques, a Karnerva model can always be implemented as a two-layer perceptron.

With this model, one still tries to minimize the same variance error, but over a set of restricted parameters (the connection weights of the top layer). Consequently, a global minimum is found with respect to this subset, whereas in the back-propagation case, only a local minimum is found with respect to all the possible connection weights.

The following section presents the algorithm we used for our purposes, which can be seen as a generalization of these ideas, with a somewhat different approach.

2.6 Randomly Linearized Non-linear Neural Networks.

The randomly linearized adaptation algorithm follows the idea of a much faster straightforward algorithm (which amounts to a linear least mean square adaptation after fixed non-linear transformation) versus a slower adaptation of all parameters of the representation (see figure 5). This kind of procedure (fix the first layer, adapt on the second layer) has an obvious apparant drawback of minimizing the mean square error only on a subset of all possible parameters, but this is compensated by the ability to chose a very large number of non-linearities (up to 1,000 in our simulations). This high dimensionality implies *a priori* the ability to find a minimium of error closer to the global minimum, and allowed us to solve the non-linear adaptation problem by a linear one (namely least mean square error adaptation).

Of course one has to have some insight about how any given mapping is decomposed into non-linearities in order to make this procedure successful by setting the connections weights of the first layer at the begining of the training algorithm. For our simulations, we have chosen to set all these parameters to random values in the approximated range [–1;+1]. This "randomly linearized adaptation" provides excellent estimations in our simulations, and was not improved by a post-processing of any kind of back-propagation algorithm. Moreover, the representation scheme is as described in figure 3.2 and a linear + constant part was adapted as well.

In order to prove the completeness of the decomposition used in this algorithm, one has to prove that the following variance of the error tends to 0, for large training sets, as the number NL of non-linearities tends to infinity.

$$\mathbf{E} = \sum_{\tau} \sum_{j} (O_j^{\tau} - \sum_{k=1}^{NL} \mathbf{A}_{kj} H_k^{\tau})^2 \tag{19}$$

τ is the index associated with the training set, and H_k^{τ} is the output of the k^{th} intermediate non-linear node for the τ^{th} input pattern of the training set.

E can be rewritten for a given training set output O_j as the squared norm of $\underline{O} - \mathbf{H}.\underline{\mathbf{A}}$, where **H** is the matrix of elements $(H_k^{\tau})_{\tau,k}$. The least mean square solutions for the $\mathbf{A}_{kj}$'s is:

$$\underline{\mathbf{A}}_j = ({}^t\mathbf{H} \,.\, \mathbf{H})^{-1}({}^t\mathbf{H} \,.\, \underline{O}). \tag{20}$$

where ${}^t\mathbf{H}$ denotes the transpose of **H**. The training set error obtained is:

$$| \underline{O} - \mathbf{H}\,({}^t\mathbf{H} \,.\, \mathbf{H})^{-1} \;{}^t\mathbf{H} \,.\, \underline{O} \,|^2 \tag{21}$$

We conjecture that for a large class of continuous mappings, the operator $\mathbf{H}({}^t\mathbf{H}.\mathbf{H})^{-1}\,{}^t\mathbf{H}$ converges to the identity matrix (of dimension the number of elements in the training set) as NL tends to infinity, provided that the fixed values for the first layer's connection weights are homogeneous, in some sense, and that the training set samples are independant, with $N \gg NL$. In other words, these values don't influence the result as far as the adaptation of a given smoothed mapping is concerned.

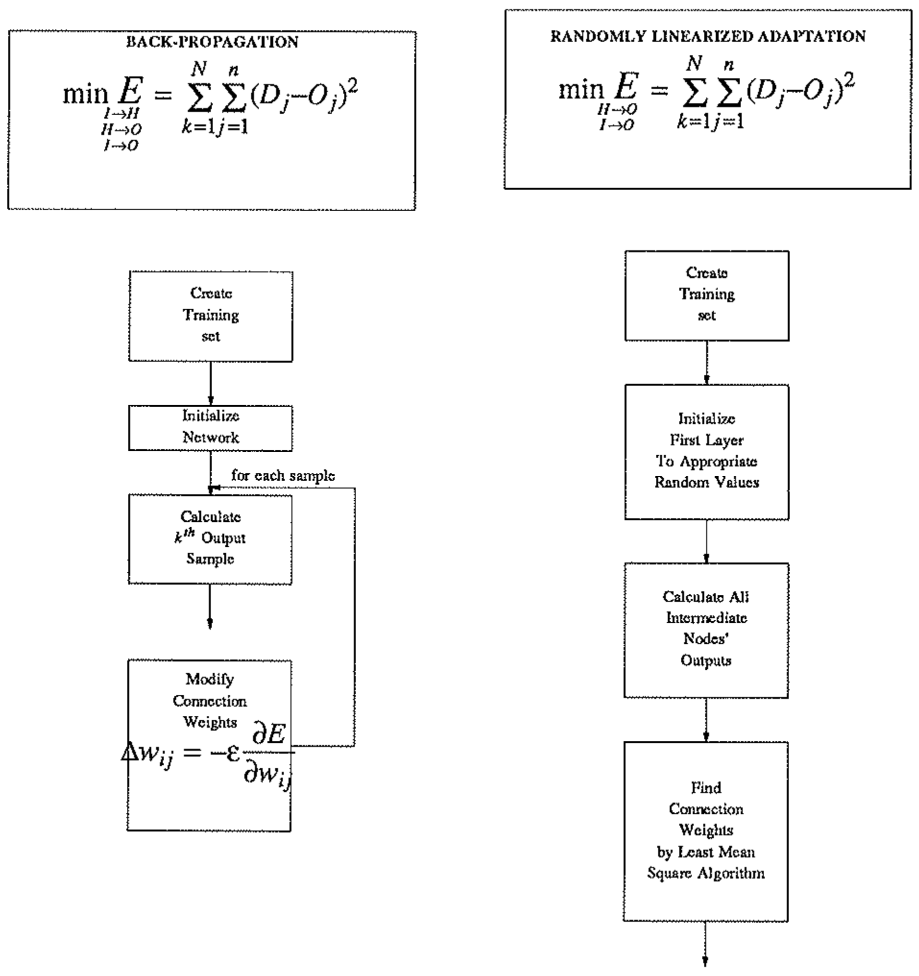

BACK-PROPAGATION
$\min_{I\to H,\ H\to O,\ I\to O} E = \sum_{k=1}^{N}\sum_{j=1}^{n}(D_j-O_j)^2$
Create Training set
Initialize Network
for each sample
Calculate k^{th} Output Sample
Modify Connection Weights
$\Delta w_{ij} = -\varepsilon\frac{\partial E}{\partial w_{ij}}$
RANDOMLY LINEARIZED ADAPTATION
$\min_{H\to O,\ I\to O} E = \sum_{k=1}^{N}\sum_{j=1}^{n}(D_j-O_j)^2$
Create Training set
Initialize First Layer To Appropriate Random Values
Calculate All Intermediate Nodes' Outputs
Find Connection Weights by Least Mean Square Algorithm

Figure 5. Comparison of two two-layer perceptron algorithms: Back-Propagation and Randomly Linearized Adaptation.

Saturation probability and scaling.

It is possible to estimate the average saturation of the non-linear intermediate nodes under a simple assumption on the weights' random number generator, in order to predict reasonable scaling normalization for the network. Assuming the $\mathbf{R}_{lk}$ are independant gaussian random variables of mean 0 and variance σ, the linear combinaison $\sum_{l=1}^{n}\mathbf{R}_{lk}I_l + \mathbf{r}_k$ feeding each non-linearity ϕ has a centered gaussian density of probabilty of variance $\gamma = \sigma\sqrt{1+|\underline{I}|^2}$. If we assume $\phi(x) = \tanh(x)$, which is a reasonable choice, then the output of each non-linearity has the distribution:

$$DSP(H_k) = \frac{e^{-\frac{Artanh(X)^2}{2\gamma^2}}}{\gamma\sqrt{2\pi}(1-X^2)} \tag{22}$$

$$\text{for } -1 \leq X \leq +1 \,.$$

This density shows a high probability of saturation (on both sides of ϕ) for $\gamma > 2$, an approximate uniform density around $\gamma = 0.85$, and a high probability for the inputs to pass thru the linear part of ϕ for $\gamma < 0.3$ (see figure 6).

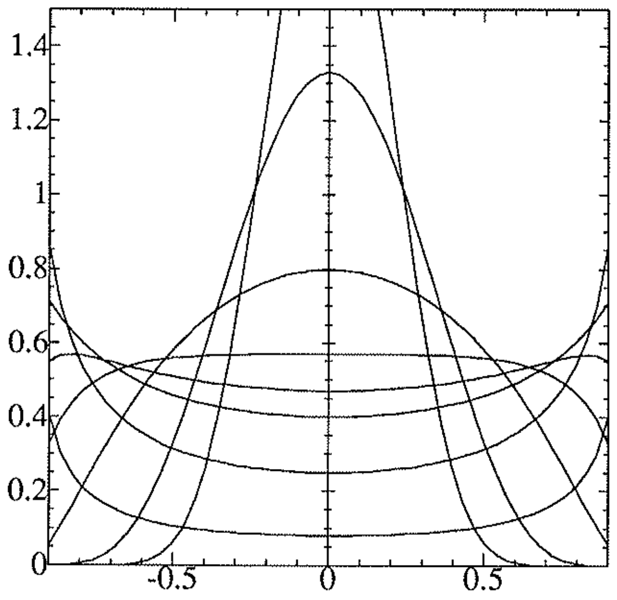


Figure 6. Density of probability DSP(H_k) of the non-linear nodes' outputs for $\gamma = \sigma\sqrt{|\underline{I}|^2+1}$ increasing from 0.2 (sharp "bump") to 5.0 (saturation on both sides)

It is therefore possible (and reasonable) to choose σ and to normalize the input to a given interval, say $[-1;+1]$ in order to avoid numerical problems of saturation or instability. This apparent restriction doesn't imply any loss of generality, since a very simple scaling procedure can be applied for 'real' inputs in the range $[I_{\min};I_{\max}]$: One simply has to change:

$$\mathbf{R}_{lk} \quad in \quad \frac{2\mathbf{R}_{lk}}{I_{\max}-I_{\min}} \tag{23}$$

$$\mathrm{r}_k \quad in \quad \mathrm{r}_k - \frac{I_{\max}+I_{\min}}{I_{\max}-I_{\min}} \sum_{l=1}^{n} \mathbf{R}_{lk}$$

Moreover, one can integrate the estimation of the constant and linear part in the non-linearities by choosing different appropriate σ's for the bottom layer's connection weights.

3. Determining Articulatory Parameters from Acoustic Parameters.

In order to have more insight on human speech production, it would be useful to provide some efficient way to determine articulator positions in the vocal tract. Until now, it remains extremely difficult to determine these articulatory parameters due to the wide range of values thay can have, and also due to a compensation mechanism they can follow, thus allowing different vocal tract shapes to provide the same speech pattern (see the discussion about the question of invertibility below). Present methods of determining measurements of articulator positions during production of speech are mainly based on x-ray and optical techniques. Therefore, very few data is available and a model was used to produce reasonable data. One way to determine articulatory data from speech is to use a synthesized speech adaptation on a continually adjusted set of glottal and vocal tract parameters[9]. The Mermelstein model[8] used in our simulations produces some vocal tract parameters that, along with other glottal parameters such as the lung pressure, are needed to drive an articulatory speech synthesizer[10], which thus mimics the way we produce speech by combining a sequence of articulatory configuration to generate a desired speech signal.

The Mermelstein model, when coupled to an acoustic model of the vocal tract, allows us to compute acoustic parameters from articulatory positions in the vocal tract via a vector of 20 areas distributed along the vocal tract: The articulator positions are used to approximate the cross sectional area of the vocal tract at 20 discrete points; each such section is then defined as the perpendicular area to a center line running through the tract, bounded by the walls of the oral cavity.

Using this model, it was therefore easy to provide training sets for training a network representation of the inverse relationship (acoustics to articulatory). The influence of the nasal tract was not considered in our present simulations nor was any modeling of glottis exitation. The acoustic parameters are directly derived from the transfert function of the vocal tract, predicted by the models used.

The question of Invertibility.

Although one given source of exitation and shape of areas produces a unique acoustic ouput (i.e. the transformation areas $\rightarrow$ acoustics is injective), it is somewhat difficult, as far as the present understanding of acoustic speech production is concerned, to determine under what circumstances the assumption of invertibility of the mapping from articulatory to acoustics fails.

At most, the representation used in our simulations seemed to prove the quasi-uniqueness of the transformation given by the (simplified) model, for our data.

Generation of the Training set.

In order to avoid the implementation of a training which involves enormous amounts of data, our training set was taken as a so-called Articulatory Codebook[7]. This codebook was merely generated by interpolating in the articulatory space different root shapes of the vocal tract, and then interpolating the interpolations. This training set is sufficiently complete (about 10,000 points) and contains association pairs of input-output (articulatory-acoustics) relatively constrained (only a subset of all possible configurations) but related to speech production, to provide a good generalization ability for the trained network.

The Mermelstein model sets position variables to the different structures of the vocal tract (pharynx, nasal and oral cavity). These positions are either indicated in fixed coordinates or relative to the structure to which they are attached. The articulatory parameters variable in our simulations were:

1. Jaw Angle (in radians): Its position is related to a fixed point and indicated by the lower teeth.
2. Tongue Center: The body of the tongue is represented as a circle of fixed radius 2 cm, of which the center is movable.
3. Tongue Tip: Rotates about a point located on the tongue body tangent to the tongue circle. Its coordinates are refered to this point. Tip and center can be considered relatively independant and can move in all directions.
4. Lip Position: The lower lip is refered to the jaw angle and can significantly change the length and shape of the vocal tract.
5. Hyoid Position: Close to the glottis where the vocal tract shape significantly changes.

The last four parameters were measured in x and y coordinates, in centimeters (see appendix). Other Mermelstein articulatory parameters were set to their nominal values: The velum area, allowing a coupling of the nasal sidebranch to the oral tract, was set to 0. The Tongue radius was also constant.

3.1 The Randomly Linearized Adaptation case.

The different estimations are represented as estimated vs. actual plottings in the appendix. Half of the articulatory codebook was taken as the training set, while the other half was used to test the network after training. The maxima of errors can be estimated in the articulatory space, as shown in the following table. However, a more understandable interpretation of these errors can be performed back in the acoustics' space, comparing the test set's inputs and acoustics computed by the model from the network-estimated outputs. The root mean square error of the logarithmic spectral distance was computed, showing that most (83%) of the test sample's estimations were under 4 dB spectral error.

Parameter	Range	Max Error
Jaw Angle	0.29-0.36 rad.	0.035 rad. (2 deg)
X Tongue Center	6.0-8.5 cm.	2.0 mm.
Y Tongue Center	3.7-6.3 cm.	3.0 mm.
X Tongue Tip	7.5-13 cm.	10.0 mm.
Y Tongue Tip	2.0-5.5 cm.	15.0 mm.
X Lip Position	0.2-1.2 cm.	3.0 mm.
Y Lip Position	-0.05-0.4 cm.	1.6 mm.
X Hyoid Position	6.1-6.4 cm.	1.4 mm.
Y Hyoid Position	8.45-9.0 cm.	2.1 mm.

TABLE 1. Estimation of Articulatory Parameters from Acoustics with Randomly Linearized Non-linear Networks.

3.2 Comparison Randomly Linearized Adaptation - Back-propagation.

In order to estimate the capabilities of both algorithms, simulations on estimation of four line spectral frequencies from fourth order reflexion coefficients were made; figure 7 shows the behaviour of both algorithms running on the same data. The training set consists of 5,000 reflexion coefficients - line spectral frequencies pairs, computed from real speech. The network is tested on another set of 1,000 samples.

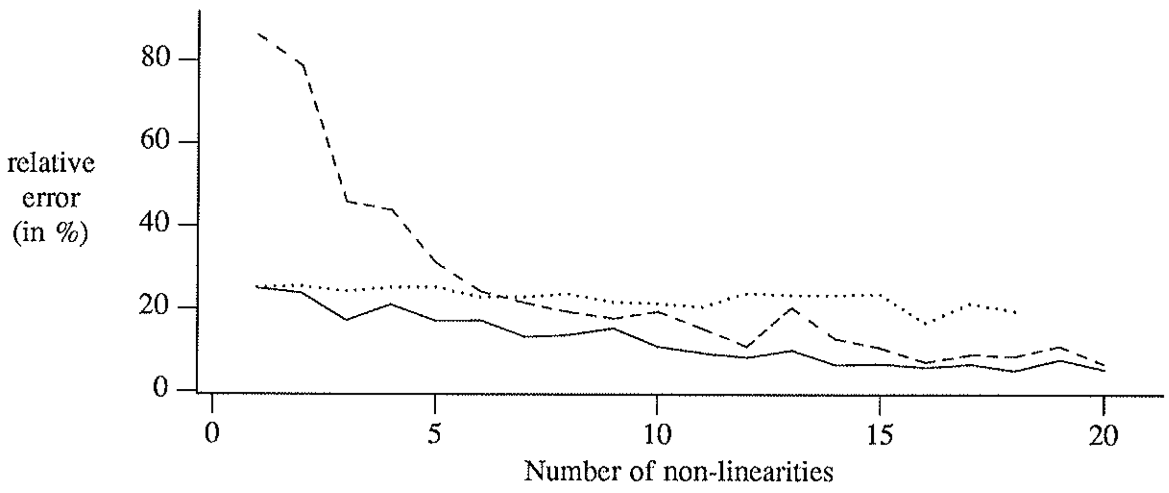


Figure 7. Relative error $\sqrt{\frac{E}{|\underline{o}|^2}}$ versus NL (number of non-linear nodes) for different algorithms running on a 2-layer perceptron: back-propagation with linear part (dotted), randomly linearized adaptation with linear part (solid), randomly linearized adaptation without linear part (dashed).

The mapping involved in this estimation is far from being simple: In order to determine the exact values of line spectral frequencies from reflexion coefficients, one needs to compute the linear prediction polynomial $P(z)$ from reflexion coefficients using the Levinson-Durbin recursive method, then compute the complex roots of the polynomials $P(z)+z^{-5}P(z^{-1})$ and $P(z)-z^{-5}P(z^{-1})$. These roots lie on the unit circle and are symetric from the real axis; the line spectral frequencies are then defined as the positive angles of these roots. A two-layer perceptron can perform this operation much faster, at least for inputs of the same nature (e.g. range) as those of the training set.

Not surprisingly, the back-propagation algorithm is slower by typically 2 or 3 orders of magnitude compared to the randomly linearized adaptation algorithm; we have thus restricted ourselves to simulations where the number of non-linear nodes was varying up to 20. This is a well known limitation for back-propagation: indeed, the randomly linearized adaptation model allow us to perform estimation with a much larger number of non-linear nodes.

Both algorithms were ran from the same initial random set of connection weights; no particular optimization procedure was investigated: the goal was to run the two algorithms independently, without any knowledge from each other. In our example, the back-propagation method gives a poorer estimation. It shows that the randomly linearized adaptation model is able to find a reasonable local minimum of the quadratic error with respect to the whole set of connection weights.

Note that the linear part can be estimated quite well using only non-linear nodes, provided that there are enough connections in the network: this is due to the large number of smaller connection weights of the first layer, that integrate the linear estimation into the linear caracteristic (near 0) of the non-linear transfert function ϕ.

4. Notes on Simulations

We ran our simulations on a Cray X-MP to speed up convergence to an exact minimum for simulations using randomly linearized adaptation and back-propagation involving real data. In addition, a conjugate gradient minimizing procedure, programmed in language C in [11], instead of the commonly used steepest descent procedure, was investigated. This is supposed to speed up the convergence by some orders of magnitude, but still was slow compared to the randomly linearized adaptation model.

ACKNOWLEDGEMENTS

The authors would like to thank S. Parthasarathy, Naftali Tishby, Mohan Sondhi, Roberto Pieraccini, Pierre de La Noue, Jean François Leber, Daniel Broseta, Ingrid Daubechies for frutful discussions, Juergen Schroeter for remarks, critics and support, Jerry Larar for simulation programs, Barabara Caspers for useful advises on UNIX use.

REFERENCES

[1] A. Lapedes, R. Farber, Non Linear Signal Processing using Neural Networks *Preprint from Los Alamos National Laboratory*, July **1987**.

[2] R.W. Prager, F. Fallside, The modified Kamerva Model for Automatic Speech Recognition. Engineering Department - Cambridge University, **1988**.

[3] Y. Bengio, R. De Mori, Use of neural networks for the recognition of place of articulation. *I.C.A.S.S.P. New York*, April 11-14, **1988**.

[4] B.S. Atal, Towards determining articulator positions from the speech signal *Speech Communication Seminar - Stockholm.* Aug. 1-3, **1974**

[5] B.S. Atal, O. Rioul, *in preparation*, October **1988**.

[6] R.J. Caballero, J.Schroeter, A graphics interface to a model of speech production - *AT&T Internal Memorandum*, Aug. 23, **1988**

[7] J.N. Larar, J. Schroeter, M. M. Sondhi, Vector-quantization of the articulatory space, *IEE Trans. Acous. Speech and Sig. Proc.*, ASSP-36, December **1988**.

[8] P. Mermelstein, Articulatory model for the study of speech production, *J. Acous. Soc. Am., 53(4), pp. 1070-1082, (***1973**).

[9] J.Schroeter, J.N. Larar, M.M. Sondhi, Speech Parameter Estimation Using a Vocal Tract/Cord Model, *Paper 8.6, Porc. ICASSP-87, vol 1, pp 308-311* (**1987**).

[10] M.M. Sondhi and J.Schroeter, A Hybrid Time-Frequency Domain Articulatory Speech Synthesizer, *IEEE Trans. Acous. Speech and Sig. Proc., ASSP-35, pp. 955-967,* (**1987**).

[11] W.H.Press, B.P.Flannery, S.A.Teukolsky, W.T.Vetterling, Numerical Recipes in C - *Cambridge University Press*, **1988**.