Neural tree density estimation for novelty detection

Dominique Martinez

Abstract— **In this paper, a neural competitive learning tree is introduced as a computationally attractive scheme for adaptive density estimation and novelty detection. The learning rule yields equiprobable quantizations of the input space and provides an adaptive focusing mechanism capable of tracking time-varying distributions. It is shown by simulation that the neural tree performs reasonably well while being much faster than any of the other competitive learning algorithms.**

I. Introduction

Identifying an input as "novel" or "abnormal" is an important task for signal segmentation or failure detection. From a statistical point of view, novelty detection is based on a sequential likelihood-ratio test between two possible statistical situations. The test relies on a sequence $\{\mathbf{x}(t)\}_{t=1}^{n}$ of independent, identically distributed, multivariate, random variables [1]. Usually, a Gaussian parametric model is assumed for the distribution. Thus, the task reduces to detecting a change in the mean parameter or the covariance matrix. By way of example, speech signals modeled by an autoregressive model can be automatically segmented by assuming Gaussian residuals [2]. However, as a drawback, the parametric function chosen might not be a good representation of the true probability density function. If no assumption is made about the form of the distribution, perhaps the simplest nonparametric technique for estimating the density consists of building up a multidimensional histogram. This can be done by partitioning the sample space into N discrete bins. Inside each bin, the density is estimated as the fraction of data points falling in that bin, normalized by the bin volume. Unfortunately, a prohibitive number of data points is required in high dimensional spaces to avoid having mainly empty bins. In both variable kernel and K-nearest-neighbor methods, the density is represented by a linear combination of basis functions located on each training data point. Because the size of the model grows with that of the training set, large computer storage and a great deal of computations are usually required to estimate the density.

Neural network techniques, closely related to Gaussian mixture models, [3] have been developed to overcome some of the disadvantages associated with the above nonparametric methods. Unfortunately, the training process, which leads to the Expectation-Maximization (EM) algorithm [4], is computationally intensive, particularly in spaces of high dimensionality. To overcome this problem, one may choose to make independent the components of the input vector prior to density estimation [5] or to reduce the number of parameters to be estimated. In the case where the mixture components share a common, fixed diagonal covariance matrix with equal elements and are assumed to have the same mixing proportion, the EM algorithm reduces to the well-known K-means clustering algorithm [6, 7]. The standard, unsupervised, competitive learning algorithm can be viewed as a sequential update version of the batch iterative K-means clustering procedure. Competition amounts to nearest-neighbor classification and results in a Voronoi or Dirichlet partition of the input space into N partition cells S_l, $l = 1 \cdots N$. However, to use competitive learning algorithms for density estimation, the probability of each neuron winning the competition needs to be estimated from the data. In addition, some neurons may never win and, therefore, never learn (dead unit). This can be avoided provided the learning rule attempts to maximize Shannon's entropy. In that case, the neurons win the competition with an equal probability $\frac{1}{N}$. The density can be estimated as $p_l \approx \frac{1}{N\,V(S_l)}$ in each partition cell S_l of volume $V(S_l)$. Several competitive learning algorithms have been developed around the principle of equiprobable quantization [8, 9, 10, 11]. However, for density estimation, this approach suffers the serious drawback of computing the volume of the Voronoi cells. Another algorithm, proposed in [12], considers a lattice of quadrilateral partition cells. Unfortunately, determining the active partition cell may be very time consuming particularly in spaces of high dimensionality, due to the lack of a simple strategy. Moreover, the learning algorithm exhibits a very slow convergence: 10,000,000 samples drawn from a uniform distribution are needed for the convergence of a 20x20 planar lattice, as pointed out in [12].

In this paper, a neural competitive learning tree is introduced as a computationally attractive scheme for adaptive density estimation. When the neural tree is continuously adapted, novelty detection can be performed on-line by comparing, at each time step, the current estimated model with an a priori model.

II. Architecture definition and initialization

The proposed approach combines the unsupervised learning property of competitive neural networks with a binary tree type structure. Like CART [13] or k-d trees [14], our procedure performs a hierarchical partitioning of the k-dimensional feature space by means of hyperplanes perpendicular to coordinates axes. This results in a binary tree structure in which each internal node i stores two scalar quantities : an index j representing the dimension orthogonal to the hyperplane and a weight w_{ij} representing the location of the hyperplane on this axis.

Laboratoire d'Analyse et d'Architecture des Systèmes (LAAS)-CNRS. 7, Av. Col. Roche. 31077 Toulouse - France

Any input vector $\mathbf{x}$ can be located with respect to this hyperplane through use of a single scalar comparison. If $x_j < w_{ij}$, the feature vector $\mathbf{x}$ is sent to the left child of node i. Similarly, the feature vector is sent to the right child if $x_j \geq w_{ij}$. The leaf nodes are associated with N disjoint partition cells S_l, $1 \leq l \leq N$. An example of hierarchical binary tree partitioning is shown in Fig. 1. The levels K of the tree are indexed so that the root located at the top has the highest level $\mathcal{L}$ and the other nodes have a level that is one lower than their father.

Let $L(i)$ and $R(i)$ be the union of partition cells belonging to the left or right subtree ($T_l(i)$ or $T_r(i)$) of node i, respectively.

$$L(i) = \bigcup_{l:S_l \in T_l(i)} S_l$$
$$R(i) = \bigcup_{l:S_l \in T_r(i)} S_l$$

The boundaries of $L(i)$ and $R(i)$ are determined by w_{ij} and the weights stored above node i.

The initialization process can be performed with N input data sampled either randomly from the training set (*random initialization*) or sequentially as data become available (*sequential initialization*) in case of an *i.i.d* sequence. The neural tree is built up by splitting nodes one at a time in order to maintain a single count in each partition cell. The cell to be further partitioned is the one in which the input sample falls. Splitting occurs in the middle of the new data point and the one previously stored. The partition hyperplane passes through the axis of the greatest gap between the two projected points. Note that the resulting neural tree is not necessarily balanced. This initialization process turns out to be the simplest and fastest. However, the overall partition of the input space depends on the N data sampled either randomly or sequentially from the training set. In order to avoid dependency on a particular set of N input samples, we might consider growing an initial neural tree by taking into account the entire training set. We refer to the following procedure as *nominal initialization*. A balanced tree is built up by splitting all existing external nodes at once while an unbalanced tree is obtained by splitting nodes one at a time. The latter implies that the node and the axis to be split must be chosen at each time step. Similar to the strategy utilized for growing tree search vector quantizers [15, 16], we consider the splitting of the node and axis contributing most to the decrease in average distortion. To reduce design complexity, cut values are chosen as the mean of the distribution, within the partition cell, projected onto the considered axis. The recursive procedure continues until the tree reaches the desired number of partition cells. It should be noted that the growing procedure is greedy because the split for each node and axis is selected without considering its subsequent effect on the tree.

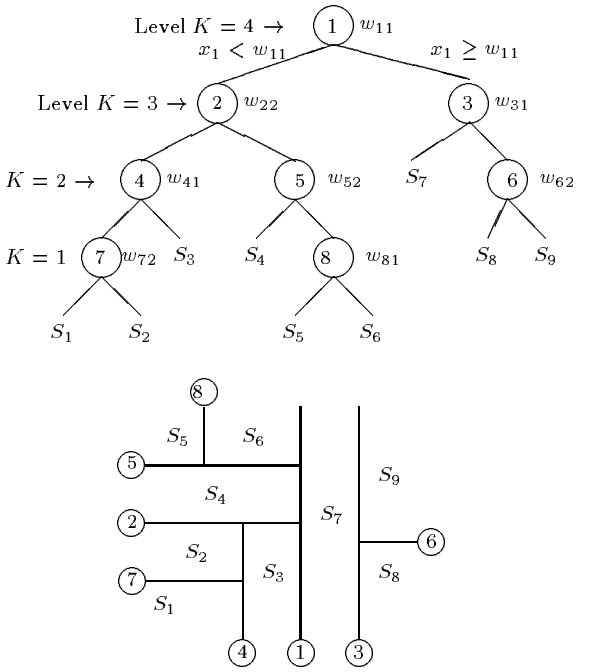

Fig. 1. Example of hierarchical partitioning over the input space induced by the binary tree. The regions $L(i)$ and $R(i)$ are determined by w_{ij} and the weights stored above node i, e.g. $L(4) = [-\infty, w_{41}) \times [-\infty, w_{22})$ and $R(4) = [w_{41}, w_{11}) \times [-\infty, w_{22})$. The root located at the top of the tree has the highest level ($K = 4$) and the other nodes have a level K that is one lower than their father.

III. Learning

We consider optimizing the weights $\{w_{ij}\}$ for a given tree topology. The proposed learning rule attempts to maximize Shannon's entropy

$$H = -\sum_{l=1}^{N} P(\mathbf{x} \in S_l) \log_2 P(\mathbf{x} \in S_l)$$

in which $P(\mathbf{x} \in S_l) \stackrel{\Delta}{=} \int_{S_l} p(\mathbf{x})\, d\mathbf{x}$. H is maximized if the input space is partitioned into N equiprobable disjoint cells, i.e. $P(\mathbf{x} \in S_l) = \frac{1}{N}$ for any l.

The following Lemma transforms the maximum entropy condition into a condition standing for every node.

Lemma 1: $P(\mathbf{x} \in S_l) = \frac{1}{N} \ \forall \ l \iff \frac{P(\mathbf{x} \in L(i))}{n_l(i)} = \frac{P(\mathbf{x} \in R(i))}{n_r(i)} \ \forall \ i$, with $n_l(i)$ and $n_r(i)$ the number of partition cells (leaf nodes) associated to the left and right subtree of node i ($T_l(i)$ and $T_r(i)$).

Proof: Given in appendix A.

To derive a suitable competitive learning rule, $\mathbb{1}_{L(i)}(\mathbf{x})$ is first defined as the code membership function of $L(i)$, *i.e.*

$$\mathbb{1}_{L(i)}(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{x} \in L(i) \\ 0 & \text{if } \mathbf{x} \notin L(i). \end{cases}$$

Likewise, $\mathbb{1}_{R(i)}(\mathbf{x})$ is defined as the code membership function of $R(i)$.

The weights $\mathbf{w} = \{w_{ij}\}$ are updated for each incoming vector $\mathbf{x}(n)$, according to

$$\begin{aligned} \Delta w_{ij}(n) &= w_{ij}(n+1) - w_{ij}(n) \\ &= \eta(n) \left(\frac{\mathbb{1}_{R(i)}(n)}{n_r(i)} - \frac{\mathbb{1}_{L(i)}(n)}{n_l(i)} \right) \end{aligned} \quad (1)$$

where $\eta(n)$ stands for the learning rate at time n. Time index n is however omitted for readability in what follows. By virtue of Lemma 1, it is easily verified that the partition's maximum entropy is equilibrium state of the system of homogeneous equations $\mathbf{E}(\Delta w_{ij}) = 0$, where $\mathbf{E}$ stands for the statistical expectation taken with respect to the input probability density function $p(\mathbf{x})$.

We first consider a *top-down learning* scheme which consists in optimizing the parameters level by level within the model. Each level of nodes is trained individually and is visited only once during training. The optimization process starts training at the highest level (root node) and goes down the tree one level at a time, holding the nodes of the upper levels fixed. Note that the top-down learning scheme is a batch algorithm with no tracking ability since the entire training set is used for training each node. Therefore, its usefulness is more theoretical than practical.

Lemma 2: Provided a continuous stationary input probability density function $p(\mathbf{x})$ and a top-down learning scheme optimizing nodes $i \in$ level K, the learning rule (1) performs a stochastic gradient descent on the positive definite cost function

$$\begin{aligned} \overline{J}(\mathrm{K}) &= \frac{1}{|K|} \sum_{i \in \mathrm{K}} \mathbf{E}\{|x_j - w_{ij}| \mid \mathbf{x} \in L(i)\} \frac{P(\mathbf{x} \in L(i))}{n_l(i)} \\ &\quad + \mathbf{E}\{|x_j - w_{ij}| \mid \mathbf{x} \in R(i)\} \frac{P(\mathbf{x} \in R(i))}{n_r(i)} \end{aligned} \quad (2)$$

with $|K|$ the number of nodes within the level K.

Proof: Given in appendix B.

Theorem 1: For a continuous, stationary, input probability density function, the top-down learning scheme converges to the partition's maximum entropy provided the learning rate meets the conditions required by the stochastic approximation theory.

Proof : The proof relies on induction. Let us consider a neural tree with $\mathcal{L}$ levels. From Lemma 2 (Eq. (7)), the condition $\frac{P(\mathbf{x} \in L(1))}{n_l(1)} = \frac{P(\mathbf{x} \in R(1))}{n_r(1)}$ is equilibrium state of $\frac{\partial \overline{J}(\mathcal{L})}{\partial w_{1j}} = 0$ for the root node $i = 1$. Assume that $\frac{P(\mathbf{x} \in L(i))}{n_l(i)} = \frac{P(\mathbf{x} \in R(i))}{n_r(i)}$ is equilibrium state of $\frac{\partial \overline{J}(L)}{\partial w_{ij}} = 0$ for any $i \in$ level $L = \mathcal{L} \cdots K$. The top-down learning scheme then optimizes nodes $i' \in$ level $K-1$ with the nodes at the levels $\mathcal{L} \cdots K$ being fixed. Likewise, $\frac{P(\mathbf{x} \in L(i'))}{n_l(i')} = \frac{P(\mathbf{x} \in R(i'))}{n_r(i')}$ is equilibrium state of $\frac{\partial \overline{J}(K-1)}{\partial w_{i'j}} = 0$. Furthermore, the optimization of $w_{i'j}$ left unchanged the equilibrium state for $i \in \mathcal{L} \cdots K$ because $L(i)$ and $R(i)$ do not depend upon $w_{i'j}$ stored at the level $K-1$. Therefore, the top-down learning scheme yields to

$$\frac{P(\mathbf{x} \in L(i))}{n_l(i)} = \frac{P(\mathbf{x} \in R(i))}{n_r(i)} \quad \forall\, i \in \mathcal{L} \cdots 1$$

and by virtue of Lemma 1

$$P(\mathbf{x} \in S_l) = \frac{1}{N} \quad \forall\, l = 1 \cdots N$$

□

Note that if the neural tree is balanced, *i.e.* $n_l(i) = n_r(i) = 2^{K-1}$ for any $i \in$ level K, and $P(\mathbf{x} \in L(i)) = P(\mathbf{x} \in R(i)) = \frac{2^{K-1}}{N}$ at convergence. Thus, the learning rule (1) minimizes the absolute error distortion measure $\mathbf{E}(|x_j - w_{ij}| \mid \mathbf{x} \in L(i) \cup R(i))$ and the weight w_{ij} converges to the conditional median of $p(x_j)$ provided $\mathbf{x} \in L(i) \cup R(i)$. Stochastic approximation considers that the learning-rate is time-varying of the form $\eta(n) \approx \frac{1}{n}$. In case of a constant learning rate $\eta(n) = \eta$, it is shown in Appendix C that, for balanced trees, the top-down learning rule converges such that the long term time-averaged cost exceeds the minimum cost by at most $\frac{\eta}{2^{K-1}N}$. Therefore, a small η is necessary to keep the time-averaged excess cost as small as desired.

To update the neural tree on-line, we transform the top-down learning scheme previously described into a *parallel learning* scheme in which all the levels are trained in parallel as the data arrive continuously. Only the parallel learning scheme will be considered for the remaining of the paper. Although Theorem 1 strictly holds for the top-down learning scheme, the convergence of the parallel learning scheme toward equiprobable quantization will be shown by simulation in the next section. However, we can have an intuitive idea about the way in which the parallel learning scheme converges. Recall that $L(i)$ and $R(i)$ are merely determined by w_{ij} and the weights stored above node i. Thus, training node $i \in$ level K with Eq. (1) does not depend upon nodes within lower levels $K' < K$. Therefore, Theorem 1 still applies for the root node. At convergence, the weight w_{1j} stored at the root will eventually fluctuate around the optimum. The variance of the fluctuations is given by $E(\Delta w_{1j})^2 = (\frac{2\,\eta}{N})^2$ for a balanced tree and therefore can be kept as small as desired since

η is a design parameter. If one assumes that η is small enough to consider w_{1j} roughly fixed, then Theorem 1 now applies for the children of the root node which converge in turn and so on. Beyond this intuitive reasoning, such a sequential convergence has been actually confirmed by simulation.

IV. Experimental results

A. Density estimation - Stationary distributions

In order to demonstrate the performance of unbalanced neural trees in terms of entropy maximization, we will consider both symmetrical and asymmetrical k-dimensional input probability density functions, for k up to 16. The entropy performance results are shown in Fig. 2 for Uniform, Gaussian and Exponential densities of unit standard deviation. The values given are averages $\pm$ standard deviations over 10 runs. For each run, the neural tree was trained on an i.i.d sequence of 20,000 samples and the entropy value was estimated using 40,000 samples. Both random and nominal initialization were considered, as discussed in section II. The number of partition cells is $N = 32\ k$ and the learning rate is linearly decreasing from $0.01\ k$ to zero over the entire training sequence. We observe that the relative difference between the entropy value obtained at convergence and the maximum entropy value is less than $3‰$ in every case. In addition, there is no significant difference in entropy performance for symmetrical or asymmetrical densities and for a random or a nominal initialization. These results clearly show the efficiency with which the neural tree converges toward equiprobable quantization.

The performance of the adaptive neural tree is now envisaged in terms of density estimation and compared with a number of unsupervised competitive learning algorithms which also attempt to maximize quantizer's output entropy: the original conscience learning (Conscience 1) [8], its slightly modified version (Conscience 2) [9], the original Frequency-Sensitive Competitive Learning (FSCL 1) [10] and its modified version (FSCL 2) [11]. Since FSCL 2 was known to approximate the density as $p(\mathbf{x})^{(3\beta+1)/(3\beta+3)}$ [11], a large β value ($\beta = 10$) was chosen for the simulations. Note that FSCL 1 is similar to FSCL 2 when β is set equal to one. For Conscience learning 1 and 2, the conscience factor was 10 and 2, as suggested in [8] and [9], respectively. The input space is partitioned, like previously, into $N = 64$ cells and the training set is generated from Uniform, Gaussian and Exponential densities $p(\mathbf{x})$ of unit standard deviation. The algorithms were iterated over the training set. The number of iterations was determined as 20,000 divided by the size of the training set. The learning rate linearly decreased to zero from 0.04 and 0.02 for the neural tree and the competitive learning algorithms, respectively. The probability density function is estimated as a multivariate histogram, constructed with

$$\hat{p}(\mathbf{x}) = \frac{1}{N} \sum_{l=1}^{N} \frac{\mathbb{1}_{S_l}(\mathbf{x})}{V(S_l)} \tag{3}$$

or as a frequency polygon, constructed by linear interpolation of the histogram. To avoid that certain partition cells may have an infinite volume, the quantization range is fixed so that the overload probability does not exceed 10^{-5}. The Mean Integrated Squared Error (MISE) is estimated by averaging the ISE $= \int |\hat{p}(\mathbf{x}) - p(\mathbf{x})|^2\ d\mathbf{x}$ over 10 runs. The MISE is plotted in Fig. 3(A)-(C) as a function of the size of the training set. We can see that, for small sample sizes ($n < 2,000$), the MISE rapidly decreases as n increases since there is insufficient data to estimate reliably the density while, for large sample sizes ($n \geq 2,000$), the MISE curve is relatively flat. As for the neural tree, Fig. 3(A)-(C) indicates that the nominal initialization outperforms the random initialization and that the frequency polygon leads to a lower MISE than the histogram. This is explained by the fact that the frequency polygon, which is inherently continuous, can approximate the continuous density better than the histogram which is discontinuous. The Uniform density constant over the input space is, however, the one situation where there is no advantage in using a frequency polygon. This is depicted in Fig. 3(A) where the difference in MISE between the histogram and the frequency polygon becomes negligible when $n = 10,000$. For every distribution, the best histogram estimation is given by FSLC2 which outperforms the other competitive learning algorithms. The histogram obtained with the neural tree has a MISE comparable to the one obtained with FSCL2 for the Exponential and Uniform densities but has a significantly higher MISE for the Gaussian density. This difference can be explained by the fact that the Voronoi polygons used as partition cells in the competitive learning algorithms are more flexible than the rectangular cells used in the neural tree. However, for the competitive learning algorithms, the estimation of the density with Eq. (3) suffers from the serious computational burden related to the computation of the volumes of the Voronoi polygons. During the simulations above, the $V(S_l)$s have been estimated by Monte-Carlo simulations using 1,000,000 random samples ! On the contrary, the computation of the $V(S_l)$s is straightforward with the neural tree and the density may be estimated on-line.

If the components of the input vectors are highly correlated, nonorthogonal partitions of the input space might offer better quantization. One can always rotate the hyperplanes with respect to the principal components of the data distribution by applying, prior to quantization, a decorrelating transformation such as the Karhunen-Loève Transform (KLT) to the original source. Figure 3(D) shows the estimated MISE for a Gaussian distribution with a full covariance matrix. The initial neural tree was built with the nominal initialization procedure. We can see that the MISE obtained by using the KLT is lower than that obtained without the KLT. However, one reason for

	Uniform		Gaussian		Exponential	
k	Random initialization	Nominal initialization	Random initialization	Nominal initialization	Random initialization	Nominal initialization
1	4.996 ± 1.1E-3	4.996 ± 8.7E-4	4.996 ± 5.9E-4	4.996 ± 9.6E-4	4.994 ± 1.1E-3	4.923 ± 3.2E-2
2	5.995 ± 8.9E-4	5.996 ± 7.9E-4	5.995 ± 1.0E-4	5.996 ± 5.8E-4	5.994 ± 1.8E-3	5.892 ± 2.1E-2
4	6.993 ± 8.0E-4	6.995 ± 9.5E-4	6.994 ± 6.1E-4	6.994 ± 6.6E-4	6.991 ± 1.3E-3	6.982 ± 5.0E-3
8	7.990 ± 7.9E-4	7.992 ± 4.0E-4	7.989 ± 1.2E-3	7.990 ± 8.0E-4	7.986 ± 1.9E-3	7.988 ± 7.0E-4
16	8.982 ± 1.3E-3	8.984 ± 1.2E-3	8.979 ± 1.2E-3	8.980 ± 1.1E-3	8.975 ± 2.1E-3	8.976 ± 9.2E-4

Fig. 2. The entries in table are entropy values ± standard deviations averaged over 10 runs. Performance is given for Uniform, Gaussian and Exponential densities of unit variance and for an input dimension k up to 16. The neural tree was trained on 20000 samples. The entropy results are estimated using 40000 samples. For each run, the number of partition cells was 32 k and the learning rate was linearly decreasing from 0.01 k to zero.

not using general hyperplanes is the increased complexity relative to the axis-parallel case. In addition, the use of the KLT requires carrying out a new estimate of the covariance matrix whenever the data distribution changes.

B. Novelty detection - Piecewise stationary distributions

Consider a sequence $\{\mathbf{x}(t)\}$, $t = 1 \cdots 20,000$, of independently distributed random variables generated from a two-dimensional parity problem in which a change occurs when $t = 10,000$. Before this change, samples are drawn from unit variance, Gaussian processes with mean vectors $(0,0)$, $(1,1)$. When $t = 10,000$, the mean vectors suddenly change to $(0,1)$ and $(1,0)$. We applied the neural tree with a fixed learning rate $\eta = 0.015$ in order to adaptively partition the input space into $N = 64$ equiprobable cells. The starting weight configuration was obtained by random initialization. Each time, the partition revealed smaller cells in high-density regions and larger cells in low-density regions relative to the current underlying input distribution. Thus, the learning rule provides an adaptive focusing mechanism capable of tracking time-varying distributions. The entropy performance of the adaptive neural tree is plotted as a time evolution in Fig. 4 and compared to those obtained with a number of unsupervised competitive learning algorithms: standard competitive learning (standard CL), Conscience 1 [8], Conscience 2 [9], neural gas learning [17], FSCL 1 [10] and FSCL 2 [11]. After extensive simulations, a fixed learning rate value $\eta = 0.04$ and $\eta = 0.015$ was chosen for the unsupervised competitive learning algorithms and neural tree 1, respectively. In addition, a linearly decreasing learning rate value, from $\eta = 0.02$ to 0 over the time intervals [1 10000) and [10000 20000), was considered for neural tree 2. Results are averages over 20 runs with different initial tree configurations obtained by random selection. We observe that standard CL, neural gas, Conscience 1 and FSCL 1 lead to very poor entropy performance. This is not surprising since standard CL and neural gas, which are stochastic gradient descent algorithms for minimizing the mean squared error distortion, are known to approximate the density as $p(\mathbf{x})^{1/3}$. Thus, they tend to over-estimate the low density regions and under-estimate the high density regions. On the other hand, the neural tree matches the input density more closely and performs at least as well as the best competitive learning algorithms such as Conscience 2 and FSCL 2.

Consider the detection of change in the distribution at $t = 10,000$. A reference tree with 64 partition cells was trained over the first 10,000 samples of the sequence. The learning rate was linearly decreased from 0.02 to 0 in order to reach correct accuracy at convergence. The latter was reached in about 10 passes over the training set for an entropy value $H = 5.9981$. Another neural tree was continuously updated over the entire sequence with $\eta = 0.015$ and a change was detected when it "differed" too much from the reference tree. Possible distance measures are the Kullback-Leibler Divergence or the log-likelihood ratio. The latter is easily estimated on-line, without knowledge of the input distribution, by calculating the volume of the active cell both for the reference and the current neural tree. Note however that the log-likelihood ratio could not be estimated on-line by using the competitive learning algorithms since it would imply computing the volume of the Voronoi cells at each time step. The behavior of the log-likelihood ratio estimated with the neural tree is plotted in Fig. 4. As expected, the change at $t = 10,000$ appears as a change in the drift of the log-likelihood ratio and can be detected by an appropriate threshold.

The last simulation explores the ability to change the overall structure of the partition by sequential initialization after detecting a change in the data distribution. Samples are drawn from a Gaussian distribution with a diagonal covariance matrix. The variance is significantly higher along the x_1-axis ($\sigma_1 = 10\ \sigma_2$) for $t < 10,000$ and along the x_2-axis ($\sigma_2 = 10\ \sigma_1$) for $t > 10,000$. Like before, the input space is partitioned into 64 cells and the neural tree is trained with a fixed learning rate $\eta = 0.015$. The starting weight configuration was obtained by sequential initialization. As explained previously, the likelihood ratio is estimated on-line, without a priori knowledge of the input distribution, by calculating the volume of the active cell. A change in drift of the log-likelihood ratio was detected at $t = 10046$ using a threshold $\lambda = 4.0$. The tree structure was then re-initialized on-line using the next 64 samples. The sequential initialization procedure led to twice as many nodes perpendicular to the axis with most

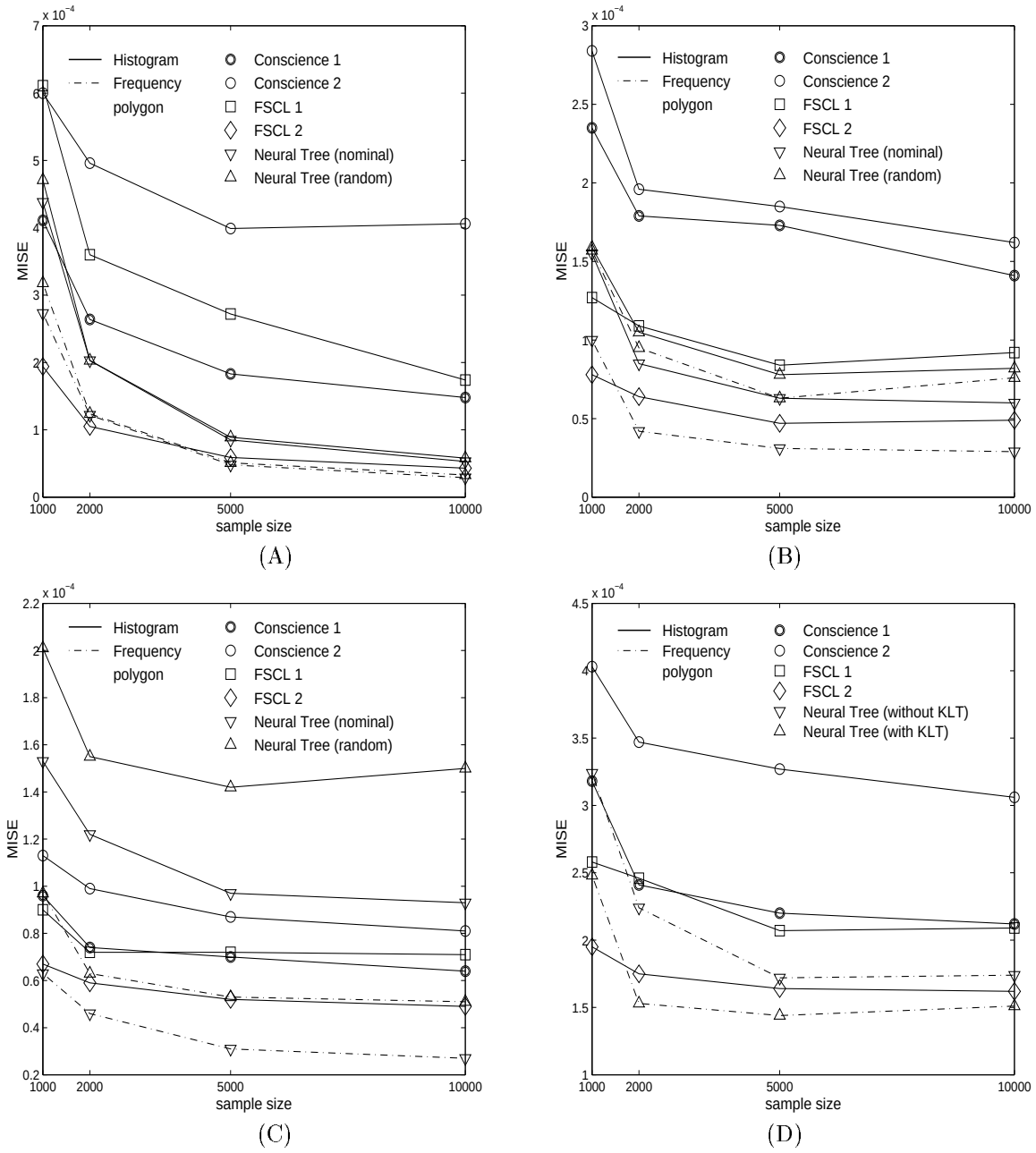

Fig. 3. Plots of the MISE as a function of the size of the training set for two-dimensional densities of unit standard deviation: Uniform distribution in (A), Exponential distribution in (B) and Gaussian distribution in (C). The Gaussian distribution in (D) has a full covariance matrix. The input space is partitioned into $N = 64$ cells. The density is estimated as a piecewise constant multivariate histogram or as a piecewise linear frequency polygon.

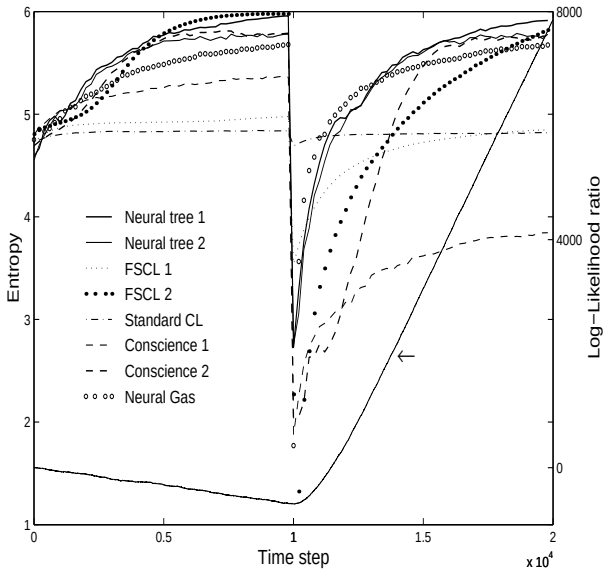

Fig. 4. Entropy performance over time (results are averages over 20 runs). The curve indicated by the arrow represents the Log-likelihood ratio estimated using the adaptive neural tree and refers to the axis on the right.

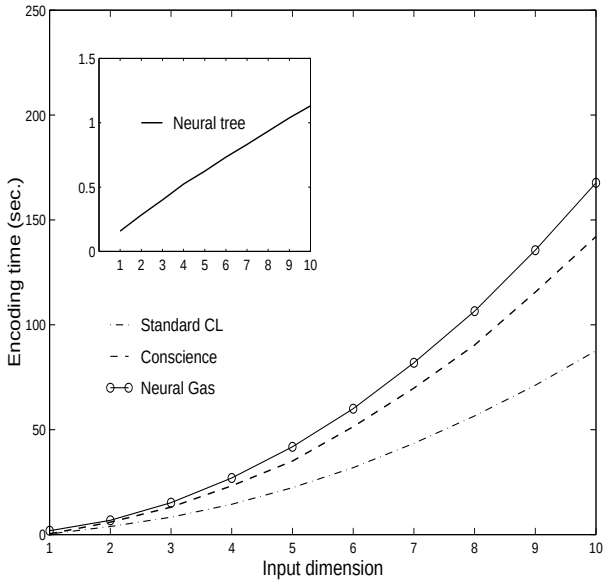

Fig. 5. Time in sec. taken to encode a set of 20 000 k-dimensional samples generated from a uniform distribution. The number of partition cells is $N = 32\ k$, for k up to 10.

of the variance, relative to the other axis. For the sake of comparison, we ran off-line a greedy bit allocation procedure similar to the one used in transform coding [18]. This algorithm allocated 5 bits to the x_1-axis and 1 bit to the x_2-axis before the change, and 1 bit to the x_1-axis and 5 bits to the x_2-axis after the change. Although non-optimal, the sequential initialization procedure was preferred on account of its simplicity and on-line capability.

C. Computational concerns

During simulation, the neural tree learning rule was at least 20 times faster than any of the other rules. A highly time-consuming part of competitive learning algorithms is due to encoding. Figure 5 compares the encoding speed of a set of $20,000$ k-dimensional input vectors for the different algorithms. The number of partition cells scales as $N = 32\ k$, for k up to 10. The encoding complexity of competitive neural networks is similar to the one of full search vector quantizers which grows exponentially with the product of bit rate r and input dimension k since the nearest-neighbor search requires $k\ N = k2^{rk}$ multiply and accumulate operations. As expected, the slowest algorithm was the neural-gas which requires ranking of all the neurons besides determining the winner. On the other hand, the neural tree is many orders of magnitude faster than the other algorithms because its encoding operation only requires comparisons, whose complexity may be assumed to be negligible. In addition, the storage requirement is reduced to $N - 1$ parameters instead of $k\ N$ for the competitive learning algorithms.

V. Conclusion

In this article, a neural competitive learning tree has been introduced as a computationally attractive scheme for adaptive density estimation and novelty detection. The algorithm performs a hierarchical clustering over the input space by means of axis-parallel hyperplanes and, thus, encoding does not require multiplication. The learning rule converges toward equiprobable quantization and provides an adaptive focusing mechanism capable of tracking time-varying distributions. If a hardware implementation of the learning rule is envisioned, a direct approach avoiding multipliers is obtained for a balanced neural tree. We are currently investigating several lines of research, including the potential for hardware implementation, the possibility of gradually changing the overall structure of the tree by splitting and merging leaf nodes, and the possibility of using general hyperplanes albeit the cost of increased complexity. The novelty detection approach proposed in this paper is being applied to the detection of changes in driving performance over time within the framework of the European program SAVE [19]. The adaptive neural tree was used to detect spectral changes in the steering angle signal under vehicle cruising conditions. Prior to detection, the signal was modeled by a third-order autoregressive model updated in a sliding window. Work is underway to take into account additional vehicle control parameters in order to build a more robust driver impairment detection system.

References

[1] M. Basseville, "Detecting changes in signals and systems–a survey," *Automatica*, vol. 24, no. 3, pp. 309–326, 1988.

[2] R. André-Obrecht, "A new statistical approach for the automatic segmentation of continuous speech signals.," *IEEE Trans.*

Acoust. Speech Signal Processing, vol. ASSP-36, no. 1, pp. 29–40, Jan. 1988.

[3] H. G. C. Travén, "A neural network approach to statistical pattern classification by semiparametric estimation of probability density function," *IEEE Trans. Neural Networks*, vol. 2, no. 3, pp. 366–377, May 1991.

[4] A. P. Dempster, N. M. Laird, and D. B. Durbin, "Maximum-likelihood from incomplete data via the EM algorithm," *Journal of the Roy. Stat. Soc*, vol. B, no. 39, pp. 1–38, 1977.

[5] L. Parra, G. Deco, and S. Miesbach, "Statistical independence and novelty detection with information preserving maps," *Neural Computation*, vol. 8, no. 2, pp. 260–269, Feb. 1996.

[6] J. B. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proc. of the 5th Berkley Symposium on mathematical statistics and probability*, vol. 1, pp. 281–297, 1967.

[7] Y. Linde, A. Buzo, and R. M. Gray, "An algorithm for vector quantizer design," *IEEE Trans. on Communication*, vol. COM-28, no. 1, pp. 84–95, 1980.

[8] D. DeSieno, "Adding a conscience to competitive learning," in *Proc. Int. Conf. on neural networks*, vol. 1, (San Diego), pp. 117–124, 1988.

[9] D. E. Van Den Bout, "TInMANN: The integer markovian artificial neural network," in *Proc. Int. Conf. on neural networks*, vol. 2, (San Diego), pp. 205–211, 1989.

[10] S. C. Ahalt, A. K. Krishnamurthy, P. Chen, and D. E. Melton, "Competitive learning algorithms for vector quantization," *Neural Networks*, vol. 3, pp. 277–290, 1990.

[11] A. S. Galanopoulos and S. C. Ahalt, "Codeword distribution for frequency sensitive competitive learning with one-dimensional input data," *IEEE Trans. Neural Networks*, vol. 7, no. 3, pp. 752–756, May 1996.

[12] M. M. Van Hulle, "Topographic map formation by maximizing unconditional entropy: a plausible strategy for "on-line" unsupervised competitive learning and nonparametric density estimation," *IEEE Trans. Neural Networks*, vol. 7, pp. 1299–1305, Sept. 1996.

[13] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and regression trees.* Belmont,CA:Wadsworth: The Wadsworth statistics/probability series, 1984.

[14] J. H. Friedman, J. L. Bentley, and R. A. Finkel, "An algorithm for finding best matches in logarithmic expected time," ACM *Trans. Math. Software*, vol. 3, no. 3, pp. 209–226, Sept. 1977.

[15] J. Makhoul, S. Roucos, and H. Gish, "Vector quantization in speech coding," *Proceedings of the IEEE*, vol. 73, no. 11, pp. 1551–1588, Nov. 1985.

[16] E. A. Riskin and R. M. Gray, "A greedy tree growing algorithm for the design of variable rate vector quantizers," *IEEE Trans. on Signal Processing*, vol. SP-39, no. 11, pp. 2500–2507, Nov. 1991.

[17] T. Martinetz, S. Berkovich, and K. Shulten, "Neural-gas' network for vector quantization and its application to time-series prediction," *IEEE Trans. Neural Networks*, vol. 4, pp. 558–569, 1993.

[18] A. Gersho and R. M. Gray, *Vector quantization and signal compression.* Boston/Dordrecht/London: Kluwer Academic Publisher, 1992.

[19] "SAVE - System for effective Assessment of the driver State and Vehicule control in Emergency situations," *http://yin.iao.fhg.de/Projects/SAVE.*

[20] A. Gersho, "Adaptive filtering with binary reinforcement," *IEEE Trans. on Information Theory*, vol. IT-30, no. 2, pp. 191–199, March 1984.

Appendix

A. Proof of Lemma 1

Here we show that the system of equalities

$$\frac{P(\mathbf{x} \in L(i))}{n_l(i)} = \frac{P(\mathbf{x} \in R(i))}{n_r(i)} \; \forall\, i \tag{4}$$

is verified iff

$$P(\mathbf{x} \in S_l) = \frac{1}{N} \quad \forall\, l \tag{5}$$

According to the definition of $L(i)$ and $R(i)$, Eq. (4) can be rewritten as

$$\frac{\sum_{l:S_l \in T_l(i)} P(\mathbf{x} \in S_l)}{n_l(i)} = \frac{\sum_{l:S_l \in T_r(i)} P(\mathbf{x} \in S_l)}{n_r(i)} \quad \forall\, i \tag{6}$$

and the condition (5) $\Rightarrow$ (4) is obvious. To prove the converse, consider first a binary tree with a single internal node i which has exactly two leaf children. Without loss of generality, (S_1, S_2) denotes the pair of partition cells associated with node i. Such a neural tree is exemplified in Fig. 6. Because $n_l(i) = n_r(i) = 1$, applying Eq. (6) gives the equality $P(\mathbf{x} \in S_1) = P(\mathbf{x} \in S_2)$ attached to node i.

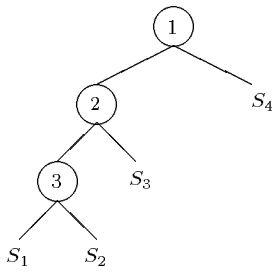

Fig. 6. Example of a neural tree exhibiting a single internal node which has exactly two leaf children.

The father of node number i inevitably has exactly one leaf child with an associated partition cell S_3. Applying Eq. (6) again leads to

$$P(\mathbf{x} \in S_3) = \frac{P(\mathbf{x} \in S_1) + P(\mathbf{x} \in S_2)}{2}$$

and therefore $P(\mathbf{x} \in S_3) = P(\mathbf{x} \in S_2) = P(\mathbf{x} \in S_1)$. Backtracking the tree to reach the root node yields $P(\mathbf{x} \in S_l) = \frac{1}{N} \quad \forall\, l$.

To complete the proof, consider a binary tree T exhibiting several internal nodes that have two leaf children. The entire tree can always be split into n disjoint subtrees $T_1, \cdots, T_n$, each one having a single internal node with exactly two leaf children, and a remaining tree T_{n+1} with the same root node than T and with the root nodes of $T_1, \cdots, T_n$ as leaves. For example, the tree shown in Fig. 1 can be split into 3 disjoint subtrees T_1, T_2, T_3 with root nodes number 4, 5 and 3 respectively, and a remaining tree T_4. Because subtrees $T_1, \cdots, T_n$ have a single internal node with exactly two leaf children, we have a set of n disjoint equalities $P(\mathbf{x} \in S_l) = \frac{1}{N}$, for all partition cells S_l associated with each subtree. These n disjoint equalities can be attached to the leaves of the remaining tree T_{n+1} which can be decomposed again as shown above to finally get $P(\mathbf{x} \in S_l) = \frac{1}{N} \quad \forall\, l$.

□

B. Proof of Lemma 2

The gradient descent algorithm requires that the cost function be differentiable at all points. Thus, it is useful to rewrite the absolute value in Eq. (2) as

$$\begin{aligned}\overline{J}(\mathrm{K}) &= \frac{1}{|K|}\sum_{i\,\in\,\mathrm{K}}\frac{-1}{n_l(i)}\int_{L(i)}(x_j-w_{ij})p(\mathbf{x})d\mathbf{x}\\ &\quad+\frac{1}{n_r(i)}\int_{R(i)}(x_j-w_{ij})p(\mathbf{x})d\mathbf{x}\end{aligned}$$

Recall that the regions $L(i)$ and $R(i)$ are merely determined by w_{ij} and the weights stored above node i which are regarded as constant in the top-down learning scheme. Since the weights stored above node i are not considered as parameters, they can be discarded from the integral limits without loss of generality.

$$\begin{aligned}\overline{J}(\mathrm{K}) &= \frac{1}{|K|}\sum_{i\,\in\,\mathrm{K}}\frac{-1}{n_l(i)}\int\cdots\int^{w_{ij}}\cdots\int(x_j-w_{ij})p(\mathbf{x})d\mathbf{x}\\ &\quad+\frac{1}{n_r(i)}\int\cdots\int_{w_{ij}}\cdots\int(x_j-w_{ij})p(\mathbf{x})d\mathbf{x}\end{aligned}$$

We take the derivative with respect to w_{ij} by making use of Leibnitz's rule. This leads to

$$\frac{\partial\overline{J}(K)}{\partial w_{ij}}=\frac{1}{|K|}\{\ \frac{P(\mathbf{x}\in L(i))}{n_l(i)}-\frac{P(\mathbf{x}\in R(i))}{n_r(i)}\ \}\qquad(7)$$

Since $\mathbf{E}\{\mathbb{1}_{L(i)}\}=P(\mathbf{x}\in L(i))$ and $\mathbf{E}\{\mathbb{1}_{R(i)}\}=P(\mathbf{x}\in R(i))$, we obtain

$$\frac{\partial\overline{J}(K)}{\partial w_{ij}}=\frac{1}{|K|}\int(\frac{\mathbb{1}_{L(i)}}{n_l(i)}-\frac{\mathbb{1}_{R(i)}}{n_r(i)})p(\mathbf{x})d\mathbf{x}$$

Stochastic approximation allows us to consider an estimation $J(K,\mathbf{x})$ of the true cost such that $\overline{J}(K)=\mathbf{E}\{J(K,\mathbf{x})\}$. Thus,

$$\begin{aligned}\Delta w_{ij} &= -\alpha\frac{\partial J(K,\mathbf{x})}{\partial w_{ij}}\\ &= \eta\ (\ \frac{\mathbb{1}_{R(i)}}{n_r(i)}-\frac{\mathbb{1}_{L(i)}}{n_l(i)}\ )\end{aligned}$$

with $\eta=\alpha/|K|$. The latter derives from the fact that $\frac{\partial\overline{J}(K)}{\partial w_{ij}}=\mathbf{E}(\frac{\partial J(K,\mathbf{x})}{\partial w_{ij}})$ because $\frac{\partial J(K,\mathbf{x})}{\partial w_{ij}}$ is piecewise continuous.

□

C. Convergence of top-down learning for balanced trees

Theorem 2: Provided a top-down learning scheme optimizing nodes $i\in$ level K in a balanced tree, the learning rule (1) with a fixed learning rate η satisfies

$$\lim_{n\to\infty}\sup\frac{1}{n}\sum_{l=1}^{n}\overline{J}_l(K)\le\overline{J}_{\min}(K)+\frac{\eta}{2^{K-1}N}$$

where $\overline{J}_l(K)$ refers to the cost attained at time l by the weight vector $\{w_{ij}(l)\}$, $i\in$ level K. The minimum cost $\overline{J}_{\min}(K)$ is attained by the optimum weight vector $\{w^\star_{ij}\}$, $i\in$ level K.

Proof: At time n, the cost function, Eq. (2), is defined by

$$\overline{J}_n(K)=\frac{1}{|K|}\sum_{i\in K}\overline{J}_n(K,w_{ij})\qquad(8)$$

with $\overline{J}_n(K,w_{ij})=\mathbf{E}\{\ u_i(n)\ |e_i(n)|\ \}$ and

$$\begin{aligned}u_i(n) &= \frac{\mathbb{1}_{R(i)}(n)}{n_r(i)}+\frac{\mathbb{1}_{L(i)}(n)}{n_l(i)}\\ e_i(n) &= x_j(n)-w_{ij}(n)\end{aligned}$$

Likewise, the minimum cost attained at the optimum weight vector is

$$\overline{J}_{\min}(K)=\frac{1}{|K|}\sum_{i\in K}\overline{J}_{\min}(K,w^\star_{ij})\qquad(9)$$

with $\overline{J}_{\min}(K,w^\star_{ij})=\mathbf{E}\{\ u^\star_i(n)\ |e^\star_i(n)|\ \}$.

From the notations above, the learning rule (1) may be rewritten as

$$\begin{aligned}\Delta w_{ij}(n) &= w_{ij}(n+1)-w_{ij}(n)\\ &= \eta\ u_i(n)\,\mathrm{sgn}\{e_i(n)\}\end{aligned}\qquad(10)$$

$$\text{where }\mathrm{sgn}(x)\ =\ \begin{cases}1 & \text{if } x\ge 0\\ -1 & \text{if } x<0.\end{cases}$$

Since Eq. (10) is related to the signed error LMS algorithm used for adaptive filtering, the convergence analysis here is adapted from the one in [20]. Let us denote $d_i(n)=w^\star_{ij}-w_{ij}(n)$ the misadjustement between the optimum weight and the current weight obtained at time n. From Eq. (10), we obtain

$$\begin{aligned}d_i^2(n+1) &= d_i^2(n)+\eta^2u_i^2(n)\\ &\quad-2\ \eta\ d_i(n)\ u_i(n)\ \mathrm{sgn}\{e_i(n)\}\\ &= d_i^2(n)+\eta^2u_i^2(n)\\ &\quad-2\eta\{d_i(n)u_i(n)+e^\star_i(n)u^\star_i(n)\}\mathrm{sgn}\{e_i(n)\}\\ &\quad+2\eta e^\star_i(n)u^\star_i(n)\mathrm{sgn}\{e_i(n)\}\end{aligned}$$

For a balanced tree, $n_r(i)=n_l(i)=2^{K-1}$ for $i\in K$ and $u_i(n)=\frac{1}{2^{K-1}}\mathbb{1}_{R(i)\cup L(i)}(n)$. From the fact that

$\mathbb{1}_{R(i)\cup L(i)}(n)$ does not depend upon $w_{ij}(n)$ and that the upper levels are fixed in the top-down learning scheme, we have $u_i^\star(n) = u_i(n)$. Therefore,

$$\begin{aligned} d_i^2(n+1) &= d_i^2(n) + \eta^2 u_i^2(n) - 2\eta u_i(n)\ |e_i(n)| \\ &\quad +2\eta e_i^\star(n) u_i^\star(n) \mathrm{sgn}\{e_i(n)\} \\ &\leq d_i^2(n) + \eta^2 u_i^2(n) - 2\eta u_i(n)\ |e_i(n)| \\ &\quad +2\eta u_i^\star(n)\ |e_i^\star(n)| \end{aligned}$$

Taking the expectation and iterating the inequality n times, we obtain

$$\begin{aligned} \mathbf{E}\{\ d_i^2(n+1)\ \} &\leq \mathbf{E}\{\ d_i^2(1)\ \} + \eta^2 \sum_{l=1}^{n} \mathbf{E}\{\ u_i^2(l)\ \} \\ &\quad +2\eta \sum_{l=1}^{n} \mathbf{E}\{\ u_i^\star(l)\ |e_i^\star(l)|\ \} \\ &\quad -2\eta \sum_{l=1}^{n} \mathbf{E}\{\ u_i(l)\ |e_i(l)|\ \} \qquad (11) \end{aligned}$$

Moreover, the top-down learning scheme assumes that the upper levels have converged and are fixed to their optimum values. Thus, we have $P(\ \mathbf{x} \in R(i) \cup L(i)\ ) = \frac{2^K}{N}$ and

$$\mathbf{E}\{\ u_i^2(l)\ \} = \frac{1}{2^{2(K-1)}} P(\ \mathbf{x} \in R(i) \cup L(i)\ ) = \frac{1}{2^{K-2}N}$$

Therefore, Eq. (11) simplifies to

$$\begin{aligned} \mathbf{E}\{\ d_i^2(n+1)\ \} &\leq \mathbf{E}\{\ d_i^2(1)\ \} + \frac{\eta^2\ n}{2^{K-2}N} \\ &\quad +2\eta\ n\ \overline{J}_{\min}(K, w_{ij}^\star) \\ &\quad -2\eta \sum_{l=1}^{n} \overline{J}_l(K, w_{ij}) \end{aligned}$$

Since the left side of the above inequality is always non-negative, we have

$$\frac{1}{n}\sum_{l=1}^{n} \overline{J}_l(K, w_{ij}) \leq \frac{\mathbf{E}\{\ d_i^2(1)\ \}}{2\eta\ n} + \overline{J}_{\min}(K, w_{ij}^\star) + \frac{\eta}{2^{K-1}N}$$

for any initial weight $w_{ij}(1)$. Since the initial misadjustement is bounded in practice, it follows that

$$\lim_{n\to\infty} \sup \frac{1}{n}\sum_{l=1}^{n} \overline{J}_l(K, w_{ij}) \leq \overline{J}_{\min}(K, w_{ij}^\star) + \frac{\eta}{2^{K-1}N}$$

and, using Eqs. (8) and (9), the long term time-averaged cost satisfies

$$\lim_{n\to\infty} \sup \frac{1}{n}\sum_{l=1}^{n} \overline{J}_l(K) \leq \overline{J}_{\min}(K) + \frac{\eta}{2^{K-1}N}$$

□