Neural Network World, 6(3):263-277, 1996.

Competitive Neural Trees for Vector Quantization

Sven Behnke
Department of Mathematics
and Computer Science
Martin-Luther-University Halle-Wittenberg
06099 Halle, Germany
behnke@informatik.uni-halle.d400.de

and

Nicolaos B. Karayiannis
Department of Electrical
and Computer Engineering
University of Houston
Houston, Texas 77204-4793, USA
Karayiannis@UH.EDU

ABSTRACT

This paper presents a self-organizing neural architecture for vector quantization, called Competitive Neural Tree (CNeT). At the node level, the CNeT employs unsupervised competitive learning. The CNeT performs hierarchical clustering of the feature vectors presented to it as examples, while its growth is controlled by a splitting criterion. Because of the tree structure, the prototype in the CNeT close to a given example can be determined by searching only a fraction of the tree. This paper introduces different search methods for the CNeT, which are utilized for training and recall. The efficiency of CNeTs is illustrated by their use in codebook design required for image compression of gray-scale images based on vector quantization.

Keywords: Neural Tree, Competitive Learning, Self-Organization, Vector Quantization, Hierarchical Clustering, Image Compression.

1. Introduction

The objective of *vector quantization* (VQ) is the subdivision of a set of vectors $\mathbf{x} \in \mathcal{R}^n$ into c subsets, or clusters, which are represented by a set of prototypes, or codewords, $\mathcal{P} = \{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_c\} \subset \mathcal{R}^n$. The finite set of prototypes $\mathcal{P}$ is often called the *codebook*. VQ can also be seen as a mapping from an n-dimensional Euclidean space into a finite set $\mathcal{P} \subset \mathcal{R}^n$. Given a codebook $\mathcal{P}$, each example $\mathbf{x}$ is represented by the prototype $\mathbf{p}_j = p(\mathbf{x}) \in \mathcal{P}$. Ideally, the function $p(\cdot)$ computes the closest prototype for every example. Consequently, each prototype $\mathbf{p}_i$ represents a cluster of examples $\mathcal{R}_i$ ($\mathbf{x} \in \mathcal{R}_i \leftrightarrow \mathbf{p}_i = p(\mathbf{x})$). These clusters $\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_c$ form a partition of the examples $\mathcal{X}$. Vector quantization is widely used in coding and compression of speech and image data.

Neural tree architectures were recently introduced for pattern classification [2, 6, 7, 8], in an attempt to combine advantages of neural networks and decision trees. By applying the decision tree methodology, one difficult decision is split into a sequence of less difficult decisions. The first decision determines which decision has to be made next and so on. From all the questions, that are arranged in a tree like structure, only some are asked in the process of determining the final answer. Neural tree architectures are decision trees with a neural network in each node. These networks perform either feature extraction from the input or make a decision. A decision must be made at internal nodes regarding the child-node to be visited next. Terminal nodes give the final answer.

Similar to neural trees is the tree structured vector quantization approach [1]. A hierarchical data structure grows while a codebook is designed. Tree structured vector quantization often utilizes binary trees. In such a case, example clusters are split in two sub-clusters and access is based on binary search.

This paper is organized as follows: Section 2 introduces the CNeT architecture and presents a generic learning algorithm. Section 3 discusses several search methods. Splitting and stopping criteria are covered in section 4. Recall procedures are described in section 5. Section 6 evaluates the performance of the CNeT in an image compression application. Finally, section 7 summarizes the results and draws conclusions.

2. CNeT Architecture and Learning

2.1. Architecture

The Competitive Neural Tree has a structured architecture. A hierarchy of identical nodes form an m-ary tree as shown in Figure 1(a). Figure 1(b) shows a node in detail. Each node contains m slots $\mathtt{s}_1, \mathtt{s}_2, \ldots, \mathtt{s}_m$ and a counter `age` that is incremented each time an example is presented to that node. The behavior of the node changes as the counter `age` increases. Each slot $\mathtt{s}_i$ stores a prototype $\mathbf{p}_i$, a counter `count`, and a pointer to a node. The prototypes $\mathbf{p}_i \in \mathcal{P}$ have the same length as the input vectors $\mathbf{x}$. They are trained to become centroids of example clusters. The slot counter `count` is incremented each time the prototype of that slot is updated to match an example. Finally, the pointer contained in each slot may point to a child-node assigned to that slot. A `NULL` pointer indicates that no node was created as a child so far. In this case, the slot is called terminal slot or leaf. Internal slots are slots with an assigned child-node.

For some splitting criteria it may be necessary to store a floating point variable `radi`. This is an estimate of the squared Euclidean distance between the prototype $\mathbf{p}_i$ of that slot and the examples for which this slot has been updated.

2.2. Learning at the Node-Level

In the learning phase, the tree grows starting from a single node, the root. The prototypes of each node form a minuscule competitive network. All prototypes in a node compete to attract the examples arriving at this node. These networks are trained by competitive learning. When an example $\mathbf{x} \in \mathcal{X}$ arrives at a node, all of its prototypes $\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_m$ compete to match it. The closest prototype to $\mathbf{x}$ is the winner. If $d(\mathbf{x}, \mathbf{p}_j)$ denotes the distance between $\mathbf{x}$ and $\mathbf{p}_j$, the prototype $\mathbf{p}_k$ is the winner if $d(\mathbf{x}, \mathbf{p}_k) < d(\mathbf{x}, \mathbf{p}_j) \; \forall j \neq k$. The distance measure used in this paper is the squared Euclidean norm, defined as

$$d(\mathbf{x}, \mathbf{p}_j) = \|\mathbf{x} - \mathbf{p}_j\|^2. \tag{1}$$

The competitive learning scheme used at the node level resembles that employed by the (unlabeled data) learning vector quantization (LVQ), an unsupervised learning algorithm proposed to generate crisp c-partitions of a set of unlabeled data vectors [4], [5]. According to this scheme, the winner $\mathbf{p}_k$ is the only prototype that is attracted by the input $\mathbf{x}$ arriving at the node. More specifically, the winner $\mathbf{p}_k$ is updated according to the equation

$$\mathbf{p}_k^{new} = \mathbf{p}_k^{old} + \alpha\,(\mathbf{x} - \mathbf{p}_k^{old}), \tag{2}$$

where α is the learning rate. The learning rate α decreases exponentially with the `age` of a node according to the equation

$$\alpha = \alpha_0 \, \exp(-\alpha_d \, \mathtt{age}), \tag{3}$$

where α_0 is the initial value of the learning rate and α_d determines how fast α decreases. The update equation (2) will move the winner $\mathbf{p}_k$ closer to the example $\mathbf{x}$ and therefore decrease the distance between the two. After a sequence of example presentations and updates, the prototypes will respond each to examples from a particular region of the input space. Each prototype $\mathbf{p}_j$ attracts a cluster of examples $\mathcal{R}_j$.

The variable $\mathtt{radi}_k$, which is an estimate of the squared Euclidean distance between the prototype $\mathbf{p}_k$ and the examples of its cluster, can be updated according to the equation

$$\mathtt{radi}_k^{new} = \mathtt{radi}_k^{old} + \alpha\,(d(\mathbf{x}, \mathbf{p}_k) - \mathtt{radi}_k^{old}). \tag{4}$$

The prototypes split the region of the input space that the node sees into subregions. The examples that are located in a subregion constitute the input for a node on the next level of the tree that may be created after the node is mature. A new node will be created only if a splitting criterion is `TRUE`.

2.3. Life Cycle of Nodes

Each node goes through a life cycle. The node is created and ages with the exposure to examples. When a node is mature, new nodes can be assigned as children to it. A child-node is created by copying properties of the slot that is split to the slots of the new node. More specifically, the child will inherit the prototype of the parent slot as well as a fraction of its radius. Right after the creation of a node, all its slots are identical. They will differentiate with the exposure to examples. As soon as a child is assigned to a node, that node is frozen. Its prototypes are no longer updated in order to keep the partition of the input space for the child-nodes constant. A node may be destroyed after all of its children have been destroyed.

2.4. Training Procedure

The generic training procedure is described below:

Do while stopping criterion is `FALSE`:

- Select randomly an example $\mathbf{x}$.
- Traverse the tree starting from the root to find a terminal prototype $\mathbf{p}_k$ that is close to $\mathbf{x}$. Let $\mathtt{n}_\ell$ and $\mathtt{s}_k$ be the node and the slot that $\mathbf{p}_k$ belongs to, respectively.
- If the node $\mathtt{n}_\ell$ is not frozen, then update the prototype $\mathbf{p}_k$ according to equation (2).
- If a splitting criterion for slot $\mathtt{s}_k$ is `TRUE`, then assign a new node as child to $\mathtt{s}_k$ and freeze node $\mathtt{n}_\ell$.
- Increment the counter `count` in slot $\mathtt{s}_k$ and the counter `age` in node $\mathtt{n}_\ell$.
- Update $\mathtt{radi}_k$ in slot $\mathtt{s}_k$ according to equation (4).

Depending on how the search in the second step is implemented, various learning algorithms can be developed. Different search methods are described in the next section. The search is the only operation in the learning algorithm that depends on the size of the tree. Hence, the computational complexity of the search method determines the speed of the learning process.

3. Search Methods

The search method determines the speed of learning and recall as well as the performance of the CNeT. A feature vector $\mathbf{x}$ constitutes the input for the search. The search method shall return a terminal prototype $\mathbf{p}_k$ that is close to the input $\mathbf{x}$. Ideally, it returns the closest terminal prototype. During learning, any terminal prototype $\mathbf{p}_j \in \mathcal{P}$ is a candidate to be selected for return. In contrast, only the prototypes that responded during learning to at least one example are candidates to be selected in the recall phase. Figure 2 shows which nodes are visited in a complete binary tree according to the search methods described in this section.

In the following, D_{tree} represents the depth of the tree, that is, the maximum number of edges on the path from a terminal node to the root. N_{tree} represents the number of terminal prototypes of the tree.

3.1. Full Search Method

The full search method is based on conservative exhaustive search. To guarantee that the prototype $\mathbf{p}_k$ with the minimum distance to a given feature vector $\mathbf{x}$ is returned, it is necessary to compute the distances $d(\mathbf{x}, \mathbf{p}_j)$ between the input vector $\mathbf{x}$ and each of the terminal prototypes $\mathbf{p}_j \in \mathcal{P}$. The prototype $\mathbf{p}_k$ with the minimum distance is returned. Note that the full search method does not take advantage of the tree structure to find the closest prototype.

The full search method is the slowest among the search methods described in this section. The time required by the full search method is of the order $O(N_{tree})$. However, the full search method guarantees the return of the closest prototype to the input vector.

3.2. Greedy Search Method

The greedy search method starts at the root of the tree and proceeds in a greedy fashion. When the search method arrives at a certain node, the distances between the input vector and all prototypes $\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_m$ of the node are computed. The prototype $\mathbf{p}_k$ with the minimum distance to the presented feature vector $\mathbf{x}$ is selected and called the winner. If a child-node is assigned to the slot $\mathtt{s}_k$ that contains the winner, then the greedy method is applied recursively to that node. Otherwise, the winner $\mathbf{p}_k$ is returned to the calling function.

The greedy method expands only one slot per level. Therefore the searched subtree is only a simple path from the root to the returned prototype. Since the time needed to expand each node is constant, the greedy search runs in $O(D_{tree})$. For trees that are balanced to some extent, the running time is of the order $O(\log N_{tree})$. The greedy search method is the fastest among the search methods described in this section and local. However, whenever a prototype is not the winner, the subtree whose root is the corresponding node is not searched. Since the greedy method will not always return the terminal prototype with the minimum distance to the presented feature vector $\mathbf{x}$, its performance is expected to be inferior to that of a CNeT trained using the full search method.

3.3. Local Search Method

The local search method is based on the idea of finding the prototype $\mathbf{p}_k$ that is closest to the input vector even though a prototype on the path from the root to $\mathbf{p}_k$ has only been the second best prototype in its node. The local search method searches a subtree that has at most i nodes on the ith level $(i = 1, 2, \ldots, D_{tree})$. The terminal prototypes in this subtree are those which have at most one prototype on their path to the root that was only second best. The local search method starts at the root of the tree. The distances between the input vector $\mathbf{x}$ and all prototypes $\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_m$ in the node are calculated. Then, the local search method is called recursively for the winning prototype. In the subtree that is below the winner, the prototypes are still allowed to loose. The greedy search method is called for the second best prototype. In its subtree prototypes are not allowed to loose once more. They have to win in order to make the search method expand them. If a winning prototype has no child-node assigned, it can not be expanded. Instead, it is returned to the calling function. The two recursive calls to the local and the greedy search method will both return a prototype that is close to the input. The closer one of the two is determined and returned. Table 1 shows the local search method in pseudocode.

The local search method uses only information that is locally available to make decisions. At most i slots are expanded on the ith level of the tree. Therefore, the time required by the local search method to return a terminal prototype is of the order $O(D_{tree}^2)$, which is usually $O((\log N_{tree})^2)$. Note that the method is more efficient for trees with a higher fan out m. The local search method returns the closest terminal prototype more often than the greedy method. Thus, the use of the local search method instead of the greedy search method is expected to improve the performance of the CNeT. Although the local search method is easy to implement and fast for small trees, its asymptotic running time is quadratic in the depth of the tree D_{tree}. If the use of a local method is not necessary for a given application, the time required for the search can be moderated by using the global search method.

3.4. Global(ω) Search Method

The global search method expands the nodes of the tree level by level, starting at the root. Expanding a node means computing the distances between all of its prototypes $\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_m$ and the feature vector $\mathbf{x}$. After this is done for all the nodes that are to expand at this level of the tree, the ω prototypes with the smallest distances are selected. If a selected prototype has a child-node assigned, this child-node will be expanded during the next expansion step. Suppose a selected prototype is a terminal prototype. If its distance to the given feature vector is the smallest so far, then the prototype is the new candidate to be selected for return. When no more prototypes are to expand, the global search algorithm terminates and returns the best terminal prototype seen. Table 2 shows the global(ω) search method in pseudocode.

The global search method searches a subtree that has a width of at most ω. Hence, the time required by the global search method to return a terminal prototype is of the order $O(\omega D_{tree})$. Clearly, the speed of this search method depends on the choice of ω. If $\omega = 1$, then the global and the greedy search methods are equivalent in terms of their time requirements. If $\omega > 1$, then the search by the global method takes longer but the probability that the search returns the closest prototype increases. Thus, the selection of ω allows the user to balance the tradeoff between the time required for the search and the performance of the CNeT. Since ω is a parameter that is not growing with the problem size and complexity, $O(\omega D_{tree})$ is not sufficiently higher than $O(D_{tree})$. In other words, the time required for the global search method grows with the problem size only as fast as the time required for the greedy search method.

4. Splitting and Stopping Criteria

The splitting criterion must be `TRUE` when the proposed leaning algorithm splits a slot and creates a new node. Meaningful splitting criteria are needed for growing balanced trees with satisfactory performance. It is also important to give each node enough time to find a good partition of its input region. For this reason, splitting of immature nodes must be avoided. The splitting criterion will be `FALSE` for nodes below the maturity age τ. The stopping criterion is used to terminate the training.

Since vector quantization is an unsupervised process, the examples are not labeled. After the tree is fully grown, each example $\mathbf{x} \in \mathcal{X}$ is represented by a similar prototype. Hence, the objective of the splitting criterion is to grow the branches of the tree that respond to examples that vary a great deal, while prohibiting growth at locations where uniform examples arrive. The variation of the examples is assessed by the variable $\texttt{radi}_k$, which maintains in each slot $\texttt{s}_k$ an estimate of the deviation of the examples that resulted in an update of the corresponding prototype $\mathbf{p}_k$. In order for this assessment to be independent

from the overall variation of the examples, the splitting criterion also involves the estimate $\texttt{radi}_0$ of the deviation of the entire set of examples $\mathcal{X}$. The splitting criterion is TRUE when the number of updates for the prototype $\mathbf{p}_k$ is greater than $\tau(1+\gamma\,\texttt{radi}_0/\texttt{radi}_k)$, where τ is the maturity age of the node and $0 \leq \gamma \leq 1$. If $\gamma = 0$, then the radius of each slot is ignored and the splitting criterion is exclusively based on maturity. The splitting criterion is increasingly influenced by the radius of each slot as γ increases from 0 to 1. The splitting criterion must be FALSE if the codebook size is fixed to c and the number of terminal prototypes N_{tree} is already equal to c.

To ensure sufficient training of the terminal nodes, the learning must continue after the tree stops growing. Additional training of the terminal prototypes of a grown tree is guaranteed by the following two stopping criteria:

- **Maturity Stop:** Stop when all nodes have reached the maturity age τ.
- **Parametric Stop:** If v is the number of example presentations that is needed to grow a tree of the desired size, then continue to train the tree for ηv more steps without further splitting. It was experimentally found that in most of the applications the value $\eta = 1/4$ can be chosen to balance the tradeoff between performance gain and training time.

5. Recall Procedures

The trained CNeT contains a representation of the examples $\mathbf{x} \in \mathcal{X}$ that have been presented to it. Its terminal prototypes $\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_c$ are representatives of the example clusters $\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_c$. The recall procedure begins with a search. For a given input vector $\mathbf{x}$, the recall procedure computes a reference to a prototype $\mathbf{p}_k = p(\mathbf{x}), \mathbf{p}_k \in \mathcal{P}$ that is close to $\mathbf{x}$. The full search method can be used in the recall phase. The other possible choice is the same search method used in the training phase. Although the second choice results in faster recall, there is no guarantee that the returned prototype $\mathbf{p}_k$ will be the one closest to $\mathbf{x}$.

The search will return a terminal prototype $\mathbf{p}_k$ which represents a cluster $\mathcal{R}_k$ that contains the input vector $\mathbf{x}$. For a given codeword $\mathbf{p}_k$, a binary reference can be generated by employing one of the following two encoding schemes:

- **Fixed Length Representation:** All codewords are labeled with a bit-string of constant length.
- **Variable Length Representation:** In a binary tree the label for $\mathbf{p}_k$ is obtained by traveling the path from the root of the tree to $\mathbf{p}_k$. If $\mathbf{p}_k$ is located in the left subtree of the actual node, then a 0 is added to the label. Otherwise $\mathbf{p}_k$ is located in the right subtree and a 1 is added to the label. This encoding scheme generates a complete description of the location of the prototype $\mathbf{p}_k$ in the tree.

The variable length representation is useful if the tree is grown in a such way that the codewords representing many examples are close to the root. In such a case, it is possible to represent frequent inputs by short references and less frequent input vectors by longer references. The generated code is a prefix code and reduces the overall codeword length.

6. Experimental Results

A natural way to apply vector quantization to images is to decompose a sampled image into rectangular blocks of fixed size and then use these blocks as vectors. The available vectors, also called training vectors, are divided into clusters, while the prototypes provide the codevectors or codewords. The image is reconstructed by replacing each image block by its closest codevector. As a result, the quality of the reconstructed image strongly depends on the codebook design [3].

This section presents an experimental evaluation of the CNeT used in the compression of real image data. Figure 3(a) shows the original Lena image of size 256×256, which was used as a test image. The image was divided into 4×4 blocks and the resulting 4096 vectors were used as the training vectors. The CNeT was used to design codebooks consisting of 2^n vectors, which implies the representation of each image block containing $4 \times 4 = 16$ pixels by n bits. In other words, the compression rate was $n/16$ *bits per pixel* (bpp). The reconstructed images were evaluated by the *Peak Signal to Noise Ratio* (PSNR), defined as

$$\mathrm{PSNR} = 10 \log_{10} \frac{255^2}{\frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N}(F_{ij} - \hat{F}_{ij})^2},$$

where 255 is the peak gray level of the image, F_{ij} and $\hat{F}_{ij}$ are the pixel gray levels from the original and reconstructed images, respectively, and $N \times N$ is the total number of pixels in the image.

A codebook of size $c = 256$ was designed using the global(3) search method. Figures 3(b) and 3(c) show the reconstructed and the difference images, respectively. As the difference image shows, the reconstructed image differs from the original image mostly at edges. Note that a gamma correction has been applied to the difference image in order to make small differences visible.

The role of the strategy employed for splitting nodes during training was investigated in these experiments by evaluating the influence on the codebook design of the parameter γ, which is involved in the splitting criterion, and the maturity age τ. It was experimentally found that the CNeT achieves a better reconstruction of smooth areas of the image if the splitting criterion is exclusively based on maturity, that is, if $\gamma = 0$. On the other hand, the CNeT achieves a better reconstruction of the image details if the splitting criterion is based on the radius of each slot, that is, if $\gamma = 1$. It was also found that the quality of the reconstructed images improved as the maturity age τ increased from 16 to 256. However, the quality of the reconstructed images was not practically affected as the maturity age increased above $\tau = 256$.

The influence of the codebook size on the quality of the reconstructed images is illustrated in Figure 4(a), which plots the PSNR of the images reconstructed from codebooks with sizes varying from 4 to 512. In these experiments, the same search method was used for both learning and recall. Clearly, the quality of the reconstructed image depends strongly on the codebook size. Regardless of the codebook size, the performance of the CNeT was only slightly affected by the search method employed.

The last set of experiments evaluated the effect of the codebook size on the computational complexity of different search methods. Because of the CNeT structure, the codebook size c relates to the size of the grown CNeT. The computational complexity of each search method is measured by the average number of evaluations of the distance function $d(\cdot,\cdot)$ in the recall phase. Figure 4(b) shows the computational complexity of different search methods when the codebook size varied from 4 to 512. The full search method grows linearly with the codebook size. The local search method grows much slower. The greedy search method has the shortest recall times, since it grows only logarithmically. The global(3) search method has some overhead for small trees, but grows only logarithmically. Consequently, for large trees it is much faster than the local search method.

7. Summary and Conclusions

This paper introduced Competitive Neural Trees, presented learning and recall algorithms, discussed different search methods, and described criteria for splitting the nodes and stopping the training. The CNeT combines the advantages of competitive neural networks and decision trees. It performs hierarchical clustering by employing competitive unsupervised learning at the node level. The main advantage of the CNeT is its structured, self-organizing architecture that allows for short learning and recall times. The performance of a trained CNeT depends on the search method employed in the learning and recall phases. Among the search methods proposed in this paper, the greedy search method achieves the fastest learning and recall to the expense of performance. On the other hand, the full search method achieves the best performance to the expense of learning and recall speed. The local and global search methods combine the advantages of the greedy and full search methods and allow for tradeoffs between speed and performance.

References

[1] A. Buzo, A. H. Gray, R. M. Gray, J. D. Markel, "Speech coding based upon vector quantization," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 28, pp. 562–574, 1980.

[2] L. Fang, A. Jennings, W. X. Wen, K.Q.-Q. Li, T. Li "Unsupervised learning for neural trees," *Proceedings International Joint Conference on Neural Networks*, vol 3, pp. 2709–2715, 1991.

[3] R. M. Gray, "Vector quantization," *IEEE ASSP Magazine*, vol. 1, pp. 4-29, April 1984.

[4] T. Kohonen, *Self-Organization and Associative Memory*, 3rd Edition, Springer-Verlag, Berlin, 1989.

[5] T. Kohonen, "The self-organizing map," *Proceeding of the IEEE*, vol. 78, pp. 1464–1480, 1990.

[6] T. Li, L. Fang, and A. Jennings "Structurally adaptive self-organizing neural trees," *Proceedings International Joint Conference on Neural Networks 92-Seattle*, vol. 3, pp. 329–334, 1992.

[7] S. R. Safavian, D. Landgrebe, "A survey of decision tree classifier methodology," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 21, pp. 660–674, 1991.

[8] A. Sankar, R. J. Mammone, "Growing and pruning neural tree networks," *IEEE Transactions on Computers*, vol. 42, pp. 291–299, 1993.

local(x)

- $\mathbf{p}_{win} := \mathbf{p}_{k_1} \in \{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_m\}$ such that $d(\mathbf{x}, \mathbf{p}_{k_1}) \leq d(\mathbf{x}, \mathbf{p}_j)\, \forall j \neq k_1$
- $\mathbf{p}_{sec} := \mathbf{p}_{k_2} \in \{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_m\}$ such that $k_2 \neq k_1$ and $d(\mathbf{x}, \mathbf{p}_{k_2}) \leq d(\mathbf{x}, \mathbf{p}_j) \forall j \neq k_1, j \neq k_2$
- $\mathtt{s}_{win}$:= the slot $\mathbf{p}_{win}$ belongs to
- **if** ($\mathtt{s}_{win}$.child $\neq$ NULL) **then** $\mathbf{p}_{win} := \mathtt{s}_{win}$.child $\rightarrow$ **local**(**x**)
- $\mathtt{s}_{sec}$:= the slot $\mathbf{p}_{sec}$ belongs to
- **if** ($\mathtt{s}_{sec}$.child $\neq$ NULL) **then** let $\mathbf{p}_{sec}$ be $\mathtt{s}_{sec}$.child $\rightarrow$ **greedy**(**x**)
- **if** ($d(\mathbf{x}, \mathbf{p}_{sec}) < d(\mathbf{x}, \mathbf{p}_{win})$) **then return** ($\mathbf{p}_{sec}$)
  else return ($\mathbf{p}_{win}$)

Table: 1: Pseudocode of the local search method.

global(x, ω)

- $\mathtt{d}_{min}$:= MAXDISTANCE
- $\mathbf{p}_{win} := \mathbf{p}_1$
- active_nodes := $\mathtt{n}_0$
- **while** (active_nodes $\neq \emptyset$)
  do
  - active_slots := $\emptyset$
  - **for all** nodes $\mathtt{n_i} \in$ active_nodes
    do **for all** slots $\mathtt{s}_j$ in $\mathtt{n_i}$
      do $\mathbf{p}_j$:= the prototype in $\mathtt{s}_j$
        if ($\mathtt{s}_j$.child $\neq$ NULL)
        then active_slots := active_slots $\cup \{\mathtt{s}_j\}$
        else if ($d(\mathbf{x}, \mathbf{p}_j) < \mathtt{d}_{min}$) **then** $\mathtt{d}_{min} := d(\mathbf{x}, \mathbf{p}_j)$, $\mathbf{p}_{win} := \mathbf{p}_j$
  - active_nodes := $\emptyset$
  - **for** the ω slots $\mathtt{s}_j \in$ active_slots that have the smallest $d(\mathbf{x}, \mathbf{p}_j)$
    do active_nodes := active_nodes $\cup \{\mathtt{s}_j$.child$\}$
- **return** ($\mathbf{p}_{win}$)

Table: 2: Pseudocode of the global(ω) search method.

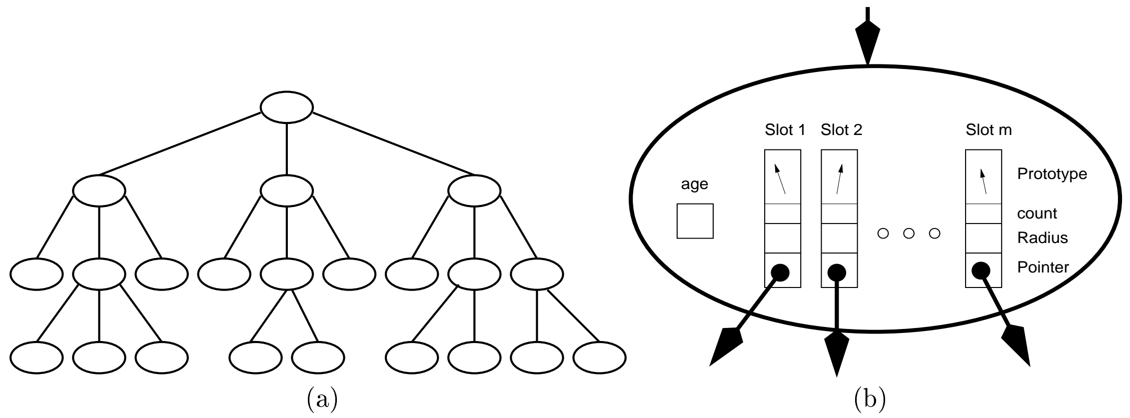


Fig. 1: The architecture of the CNeT: (a) the tree structure, (b) a node in detail for unsupervised learning.

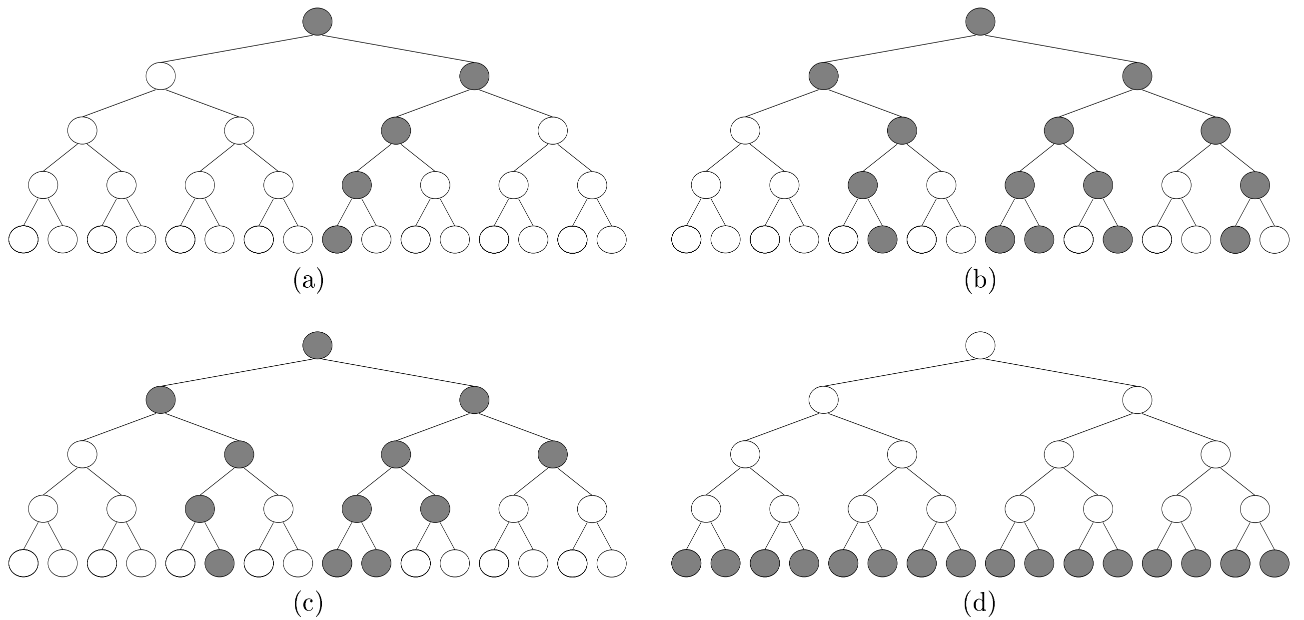
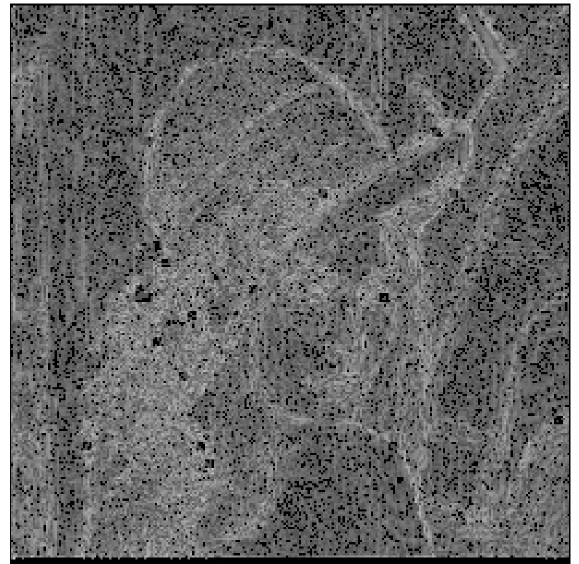


Fig. 2: Shaded nodes visited by different search methods: (a) greedy search method, (b) local search method, (c) global(3) search method, (d) full search method.

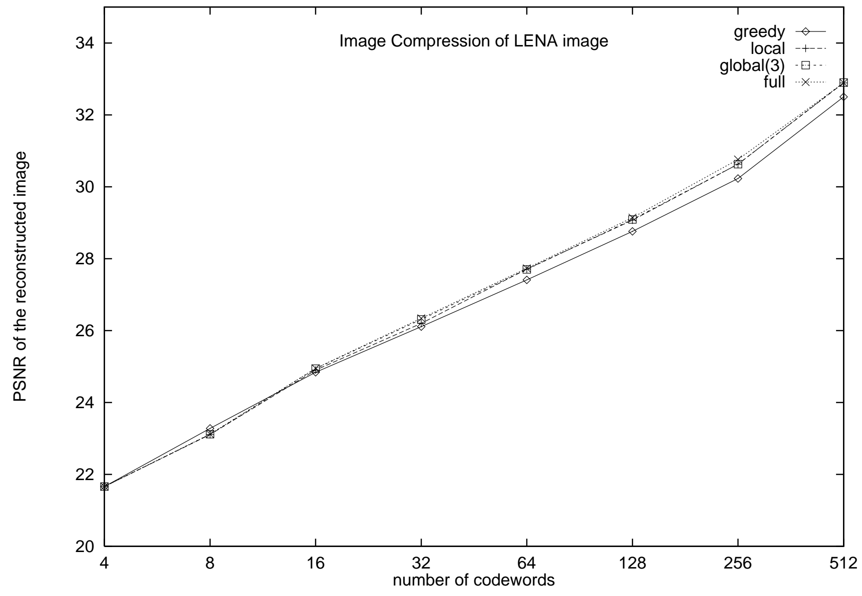
(a)

(b)

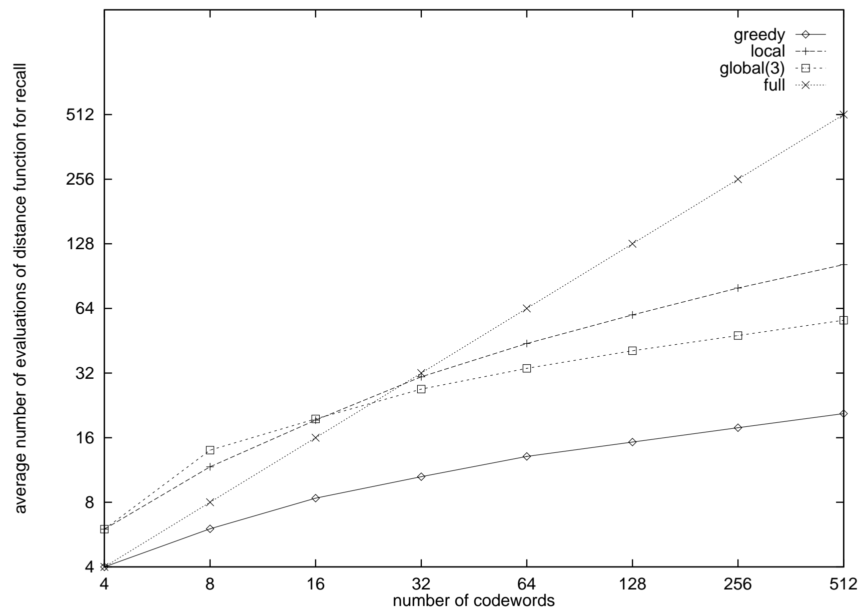


(c)

Fig. 3: Image reconstruction from a codebook of size $c = 256$ designed by a CNeT using the global(3) search method for learning and recall: (a) The original image of size 256×256 with 256 grayscales, (b) the reconstructed image, and (c) the difference image.



(a)



(b)

Fig. 4: (a) PSNR of the image reconstructed from codebooks of various sizes designed by the CNeT trained using different search methods. The same method was used for learning and recall. (b) Computational complexity of different search methods.