Learning Decentralized Goal-based Vector Quantization

Piyush Gupta[*]
Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Urbana, IL

Vivek S. Borkar[†]
Department of Computer Science and Automation, Indian Institute of Science, Bangalore, India

This paper considers the following generalization of classical vector quantization: to (vector) quantize or, equivalently, to partition the domain of a given function such that each cell in the partition satisfies a given set of topological constraints. This formulation is called decentralized goal-based vector quantization (DGVQ). The formulation is motivated by the *resource allocation mechanism design* problem from economics. A learning algorithm is proposed for the problem. Various extensions of the problem, as well as the corresponding modifications in the proposed algorithm, are discussed. Simulation results of the proposed algorithm for DGVQ and its extensions, are given.

1. Introduction

In classical vector quantization (VQ), a given stream of data vectors is statistically encoded into a digital sequence suitable for communication over or storage in a digital channel. The goal is to reduce the bit rate so as to minimize communication channel capacity or digital storage requirements while maintaining a good fidelity of the data. Even in systems with substantial channel bandwidth available, such as those employing fiber optic communication links, there is still a need to compress data because of the growing amount of information that is required to be communicated or stored.

VQ has been shown useful in compressing data that arises in a wide range of applications, most prominent are image and speech processing [4, 5]. Typically, the applications which employ VQ require large amounts of storage or channel bandwidth, and can tolerate some loss of fidelity for the sake of compression. Moreover, in many applications, VQ not only compresses data, but also reduces the time complexity of

[*]Electronic mail address: piyush@decision.csl.uiuc.edu.
[†]Electronic mail address: borkar@csa.iisc.ernet.in.

the post-processing of the data (e.g., pattern recognition in images and speech recognition).

From rate distortion theory, it can be shown that VQ can achieve better compression performance than conventional techniques which are based on the encoding of scalars, even if the data source does not have any time dependencies (e.g., the data stream consists of a sequence of independent and identically distributed random variables) [5]. However, practical use of VQ has been limited because of the prohibitive amount of computation associated with existing encoding algorithms. This is because most of these algorithms, such as the LBG algorithm [9], do batch-mode processing; that is, they need to have access to the entire *training* data set for designing the quantizer. Given the large data sets required to learn an adequate representation of the input vector space in some applications (e.g., speaker-independent speech coding), the batch-mode algorithms appear infeasible using existing technology. Hence a number of adaptive VQ algorithms have been suggested which encode successive input vectors in a manner that does not depend on previous input vectors or their coded outputs [6, 8].

Many of these adaptive algorithms are based on neural network (NN) techniques. A major advantage of formulating VQ as NNs is that the large number of adaptive learning algorithms that are used for NNs can now be applied to VQ. The two most widely used techniques are: *Kohonen's self-organizing feature maps* and the *competitive learning networks*. These schemes are (relatively) computationally cheap as they adapt the quantization map with the arrival of each new data vector, and no batch-mode processing of the data stream is required.

In this paper, we formulate a generalization of VQ that extends the application domain of VQ to distributed computing and the modeling of organizations in economics. The generalization that we consider is to (vector) quantize or, equivalently, to partition the domain of a given function such that each cell in the partition satisfies a given set of topological constraints (e.g., each cell of the partition may be required to have the *cartesian-product* property). We call this formulation decentralized goal-based vector quantization (DGVQ). We propose an adaptive algorithm for the DGVQ problem.

The main motivation for considering the DGVQ formulation is the *resource allocation mechanism design* problem from economics [7, 11]. An economic organization is modeled as a hierarchy of agents and supervisors. Each agent in the organization observes its local environment. The organization is required to take an appropriate action $f(x)$ in response to the present environment x. Since the relevant information is initially dispersed among the agents, messages need to be exchanged between agents and their supervisors. The problem is to design optimal messages for given criteria. In the economics literature, criteria based on the topological properties (e.g., size) of the message space are used. Instead, we consider criteria based on the bit complexity of the messages. For this purpose, we first reformulate the two-level hierarchy model as the DGVQ problem. We then extend the proposed DGVQ algorithm for the resource allocation problem, and for other generalizations such as when the desired-outcome function f of the organization is unknown and when the function f is time-varying.

The rest of this paper is organized as follows. Section 2 gives a brief review of VQ. Section 3 discusses the mathematical formulation of the DGVQ problem and gives two specific motivations for studying the problem. In section 4 we propose an adaptive algorithm for the DGVQ problem. Section 5 extends the proposed algorithm for the resource allocation problem and other generalizations. We conclude with some continuing work in section 6.

2. Vector quantization

VQ is a process of mapping a sequence of continuous or discrete vectors into a digital sequence. The space of the vectors to be quantized is divided into a number of regions and a reproduction vector is calculated for that region. Given any data vector to be quantized, the region in which it lies is determined and the vector is represented by the reproduction vector for that region. Instead of transmitting or storing a given data vector, a symbol which indicates the appropriate reproduction vector is used. This can result in a considerable reduction in the transmission bandwidth, though at the cost of some distortion.

More formally, a vector quantizer Q of dimension l and size N is a mapping from a vector in l-dimensional euclidean space $\mathbb{R}^l$ into a finite set C containing N output or reproduction points, called *code vectors* or *codewords*. Thus,

$$Q : \mathbb{R}^l \to C \tag{2.1}$$

where $C = \{y_1, \ldots, y_N\}$ and $y_i \in \mathbb{R}^l$ for $i = 1, \ldots, N$. The set C is called the *codebook*. Associated with N-point vector quantizer Q is a partition Σ of $\mathbb{R}^l$ into N regions or cells, σ_i, $i = 1, \ldots, N$. The ith cell is defined by

$$\sigma_i = \{x \in \mathbb{R}^l : Q(x) = y_i\}.$$

Clearly $\bigcup_i \sigma_i = \mathbb{R}^l$ and $\sigma_i \cap \sigma_j = \emptyset$ for $i \neq j$.

A vector quantizer can be decomposed into two component operations, the vector *decoder* and the vector *encoder*. The encoder $\mathcal{E}$ is a mapping which assigns to each input vector $x \equiv (x_1, \ldots, x_n)$ a symbol $\mathcal{E}(x)$ in some symbol set M, and the decoder $\mathcal{D}$ is a mapping which assigns to a symbol m in M the vector $\mathcal{D}(m)$ in the reproduction alphabet C. Thus,

$$\mathcal{E} : \mathbb{R}^l \to M \quad \text{and} \quad \mathcal{D} : M \to C \subset \mathbb{R}^l. \tag{2.2}$$

The symbol set M is often assumed for convenience to be a space of binary vectors. An element m of M is called a *quantization level* (Q-level).

The goal of a quantization system is to minimize a *distortion measure* for a given codebook size N. A distortion measure d is an assignment of a nonnegative cost $d(x, \hat{x})$ associated with quantizing any input vector x with a reproduction vector $\hat{x}$. The most convenient and widely used (as well as the one adapted in this paper) measure of distortion d is the squared euclidean distance between the input vector x and its quantized vector $\hat{x} = Q(x)$, (i.e., $d(x, \hat{x}) = (\| x - \hat{x} \|_2)^2$). The performance of a system can then be quantified by the average distortion $D = E[\, d(x, \hat{x})\,]$ between the input and the final reproduction. Given such a performance criterion, the VQ design process involves the determination of a codebook that is optimal with respect to this criterion. In general, this requires the knowledge of the probability distribution of the input data. However, in most applications, this distribution is not known beforehand, and the codebook is designed through a process called *training*. During training a set of data vectors, drawn independently according to the unknown distribution, is used to construct an optimal codebook.

2.1 Adaptive algorithms for vector quantization

A number of NN techniques have been found useful in VQ encoding and codebook design or training [6, 8]. Before discussing the training algorithms, we first consider how the VQ encoder can be formulated as a NN structure. As before, let the vectors which are to be quantized be from an l-dimensional space, and let a distortion measure $d(x, y)$ be defined on this space. Let the size of the codebook C be N, and let the codewords be $y_i, i = 1, \ldots, N$. Consider a NN with N neural units, and make the ith codeword y_i the weight vector associated with neural unit i. Given any vector x that is to be encoded, x is fed in parallel to all the N neural units. Each of these units computes the distortion between the input vector and the weight vector, $d_i = d(x, y_i)$, $i = 1, \ldots, N$. The input vector is then encoded as the symbol m, where m corresponds to the index i^* of the neural unit with the minimum distortion $d_{i^*} = \min_i d_i$.

A more fundamental benefit of formulating VQ as a NN task is that the large body of NN algorithms can now be adapted to the problem of training vector quantizers. We discuss two alternative training algorithms: the competitive learning network and the Kohonen self-organizing feature maps.

The competitive learning network

Assume that the N neural units of the NN VQ are initialized with the (random) weight vectors $y_i(0), i = 1, \ldots, N$. The training algorithm iterates a number of times through the training data, adjusting weight vectors of the neural units after the presentation of each training vector. The algorithm used to adjust the weight vectors is based on competitive learning [6, 8]; that is, the input vector x is presented to all of the neural units and each unit computes the distortion between its weight and the input vector. The unit with the smallest distortion is designated as the winner and its weight vector is adjusted towards the input vector.

More precisely, let $y_i(k)$ be the weight vector of neural unit i before the input is presented. The output z_i of the unit is computed as follows:

$$z_i(k) = \begin{cases} 1 & \text{if} \quad d(x, y_i(k)) \le d(x, y_j(k)),\ j = 1, \dots, N \\ 0 & \text{otherwise.} \end{cases} \tag{2.3}$$

The new weight vectors $y_i(k+1)$ are now computed as

$$y_i(k+1) = y_i(k) + \eta(k) \cdot (x - y_i(k)) \cdot z_i(k) \tag{2.4}$$

where η is the learning rate, and is typically reduced monotonically to zero with k.

One problem with this training procedure is that it sometimes leads to neural units which are underutilized. Various heuristics have been suggested to overcome this problem, for example, *conscience mechanism* [2] or *frequency-sensitive competitive learning* [1]. The problem, however, is rather intricate. For example, there may be dead units that do not receive any input, reflecting local minima of the cost function of the VQ. On the other hand, the primary aim of VQ is to minimize an appropriate distortion measure, not to make the utilization of neural units equal. (These goals can be somewhat conflicting.) The conscience mechanism, for example, addresses the latter goal.

Kohonen self-organizing feature maps

Another NN structure that has been widely used for VQ is the Kohonen self-organizing feature map (KSFM) [8]. The KSFM and the competitive learning network are similar; however, in the KSFM structure each neural unit has an associated topological neighborhood of other units. During the training process, the winning neural unit, as well as the neural units in the neighborhood of the winner, are updated. The size of the neighborhood is decreased as training progresses until each neighborhood has only one unit.

More precisely, let $y_i(k)$ be the weight associated with the ith neural unit, and let x be the input vector. Compute the distortion $d(x, y_i(k))$, $k = 1, \dots, N$, and let the neural unit with the minimum distortion be i^*. Also, let $R(i^*)$ be the topological neighborhood associated with unit i^*. The weight update equations are:

$$y_i(k+1) = \begin{cases} y_i(k) + \eta(k) \cdot (x - y_i(k)), & i \in R(i^*) \\ y_i(k), & \text{otherwise.} \end{cases}$$

where, as before, $\eta(k)$ is the learning rate. It is clear from these equations the KSFM structure involves more computation than the competitive learning network. At each step of the training process, the neighborhood

$R(i^*)$ of the winning node must be computed and all of the units in the neighborhood updated. However, by the use of neighborhood-based adaptation, KSFM overcomes the problem of underutilized nodes, associated with the competitive learning algorithm. (It should be kept in mind, however, that the primary aim of KSFM is to form topology preserving maps.)

3. A generalization of vector quantization

In this section we formulate a generalization of VQ and give two specific motivations for studying the problem.

Suppose we are interested in quantizing $D \subseteq \mathbb{R}^l$ with the following constraints.

- *Goal-based quantization.* Given a function $f : D \to O$, where $O \subseteq \mathbb{R}^p$ for some $p \in N$. The goal of quantization is to minimize the following average distortion:

$$E = \int_D \left(\| f(x) - f(Q(x)) \|_2 \right)^2 P(dx) \tag{3.1}$$

  where Q is the quantization map and P is a given probability distribution on D.

- *Decentralization.* Given n sets, $D_1, \ldots, D_n$, such that $D = D_1 \times D_2 \times \cdots \times D_n$. Any cell σ in the constructed codebook or partition of D, denoted Σ, is required to be of the form $\sigma_1 \times \sigma_2 \times \cdots \times \sigma_n$, with $\sigma_i \subseteq D_i, i = 1, \ldots, n$ (e.g., $n = l$ and D_i is the projection of D on the ith coordinate axis).

This problem is clearly a generalization of the VQ problem considered before—if f is the identity map, $n = 1$ and $D_1 = D$, then the given formulation reduces to the classical VQ design problem. Note that the formulation is indeed a nontrivial generalization. Had we asked for just goal-based quantization, the problem would be to quantize the range of f to, say, $\{y_i : i = 1, \ldots, r\}$, and then take the preimage of each codeword y_i as well as the borders of the corresponding Voronoi regions, to obtain a partition of D. But the problem becomes much harder, even in principle, if we bring in the second constraint, decentralization, since it may now so happen that the preimage of a codeword does not have the particular cartesian-product topology desired in the second constraint.

We refer to the given formulation as decentralized goal-based vector quantization (DGVQ). Two specific motivations for studying DGVQ are discussed in sections 3.1 and 3.2.

3.1 A distributed-computing problem

Suppose we would like to compute, for every point x in a set D, the real-valued function $\bar{f}(x)$. Suppose $\bar{f}(x)$ can be written as

$$\bar{f}(x) = f(\phi_1(x_1), \ldots, \phi_n(x_n))$$

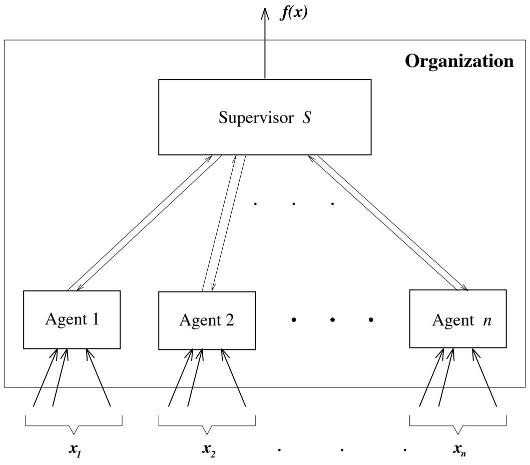


Figure 1. A two-level hierarchy model.

where $x_i \in D_i$, and $x \equiv (x_1, \ldots, x_n)$. Let $Z_i \equiv \{\phi_i(x_i) : x_i \in D_i\}$. Suppose we have $n+1$ processors. Processor i, $i = 1, \ldots, n$, receives an approximate x_i (e.g., quantized). Processor i then approximately computes $\phi_i(x_i)$ and reports the result to a central processor $n+1$ that (approximately) computes f. In other words, D_i and Z_i are partitioned into a finite number of sets; let Σ^i and σ^i denote the typical sets of those partitionings. Also, let $\Sigma \equiv \Sigma^1 \times \cdots \times \Sigma^n$ and $\sigma \equiv \sigma^1 \times \cdots \times \sigma^n$. Processor i receives the set in which x_i lies and reports to the central processor the set in which $\phi_i(x_i)$ lies. Having received these n reports, the central processor knows that $\phi(x) \equiv (\phi_1(x_1), \ldots, \phi_n(x_n))$ lies in the set $Z \cap (\sigma^1 \times \cdots \times \sigma^n)$ where $Z = \{\phi(x) : x \in D\}$. The central processor then computes its approximation of f.

We now have all the elements of the DGVQ problem. We have a partitioning Σ of Z, with its cells indexed by a set M, and which has the cartesian-product property. The problem is to find an error-minimizing Σ having the desired cartesian-product property.

3.2 Resource allocation mechanism design

In economics, an organization is modeled as a hierarchy of agents and supervisors [7, 11]. Each agent in the organization observes its local environment. The organization is required to take an appropriate action in response to the present environment. Since the relevant information is initially dispersed among the agents, messages need to be exchanged among the agents and their supervisors. The problem is to design optimal messages with respect to given criteria.

For example, consider a two-level hierarchy model composed of n agents $A_1, \ldots, A_n$, and a single supervisor S (Figure 1). Agent A_i observes its local environment x_i, which lies in a set D_i. Let $x \equiv (x_1, \ldots, x_n)$ and $D \equiv D_1 \times D_2 \times \cdots \times D_n$. The organization would like to take an appropriate *action*, namely $f(x)$ in response to the current environment x; $f(x)$ lies in some set A. f is referred to as the *desired outcome function*. Since information about x is initially dispersed among the n agents, we have to design a scheme in which some suitable communication occurs between the agents and the supervisor. In particular, we assume that the organization will use a *mechanism* on D, which we have to design. A mechanism on D is a triple $\pi = [M, (\mu_1, \ldots, \mu_n), h]$, where $M \equiv M_1 \times \cdots \times M_n$ is a set called the *message space*, with a typical element m, called a *message*; μ_i is a mapping from D_i to M_i; and h, called the *outcome function*, is from M to A. One way to interpret the operation of the mechanism is as follows: let q_i be a quantization map on D_i (i.e., q_i maps D_i to a finite subset C_i of D_i), and let l_i be an indexing map on the image of q_i (i.e., l_i maps C_i to $\{1, 2, \ldots, |C_i|\}$). Agent A_i quantizes x_i using q_i and sends the index corresponding to $q_i(x_i)$, $l_i(q_i(x_i))$, as message m_i to the supervisor S (i.e., $\mu_i = l_i \circ q_i$). Let $m \equiv (m_1, \ldots, m_n)$, $m \in M$, and l_i^{-1} be the inverse map of l_i. Then S outputs $h(m) \equiv f(l_1^{-1}(m_1), \ldots, l_n^{-1}(m_n)) \equiv f(q_1(x_1), \ldots, q_n(x_n))$ as the response of the organization to the environment x. Now we would like to find a mechanism π that minimizes the *mean-square* error E from equation (3.1) and is informationally least costly (i.e., with least communication requirement between the agents and the supervisor).

The most widely studied measures of information costs are various measures of the size of the message space M. In the bulk of the mechanisms in the economics literature both the message space and the set of outcomes are either *continua* or discrete-but-infinite [11]. But real communication and computing technologies do not in fact permit a message or an outcome to be a point of a continuum. Messages and outcomes must be quantized, and the benefits of greater precision of quantization have to be weighed against the costs (i.e., the number of Q-levels). Accordingly, it is of considerable interest to study mechanisms in which message space and outcome set are *not* continua but are, rather, quantized as in the two-level hierarchy example above.

Hence we use bit complexity of the message space M as a measure of information cost. Moreover, instead of imposing hard constraints on the bit size of the messages, we propose the following *energy* function $\mathcal{H}$:

$$\mathcal{H} = \int_D (\| f(x) - f(q_1(x_1), \ldots, q_n(x_n)) \|_2)^2 \, P(dx) + \lambda \cdot \mathcal{S} \qquad (3.2)$$

where λ is a positive weighing factor and $\mathcal{S}$ is a bit-based complexity measure of the messages. Note that the choice of λ will dictate the tradeoff between fidelity (i.e., lowering of distortion) and message complexity. Its choice perforce will involve some judgment, which makes it a "design parameter." It will be interesting to come up with a theoretically justified choice, but no obvious choice offers itself. A similar comment applies to equation (4.7) later.

In the two-level hierarchy example, $\mathcal{S}$ can be chosen as

$$\mathcal{S} = \sum_{i=1}^{n} \log_2 N_i$$

where N_i indicates the cardinality of the set $q_i(D_i)$ (i.e., the number of Q-levels for agent A_i).

4. An adaptive algorithm for decentralized goal-based vector quantization

In this section we describe an adaptive algorithm for designing the quantization maps $q_i, i = 1, \ldots, n$, such that the energy $\mathcal{H}$ from equation (3.2) is minimized. The proposed algorithm is adaptive in the sense that it tries to improve its performance with every instance of the input-output pair seen.

The basic idea is as follows. The algorithm starts by assigning a fixed number of Q-levels, say 1, to each agent. It adapts $q_i, i = 1, \ldots, n$, using a Kohonen-like rule for a fixed number, say γ, of input-output pairs seen. Concurrently, it collects the error statistics for each agent. Based on the error statistics, the algorithm adds (deletes) a Q-level in q_{i^*}, where i^* is the agent which induced maximum change in $\mathcal{H}$ in its previous adaptation. It again adapts q_is for γ input-output pairs. The algorithm repeats these two steps of adaptation and addition (deletion) of Q-levels until the energy $\mathcal{H}$ stops decreasing.

We now discuss in detail each of the steps involved in the algorithm.

4.1 Adaptation rule for single-agent case

Since we are interested in an adaptive algorithm, we seek a rule which updates the quantization maps q_i after every input-output pair $(x, f(x))$ seen, so as to minimize the energy function $\mathcal{H}$. In order to concentrate on this step of the algorithm, we consider the following simplification of the problem: Given m instances of the input-output pairs $(x(k), f(x(k))), k = 1, \ldots, m$, where $\{x(k)\}$ are independent and identically drawn from D according to the probability distribution P. Also given is a fixed number N. Consider a two-level hierarchy in which a single agent A reports to supervisor S. The quantization map q of agent A has exactly N Q-levels. The problem is to adapt the map q, keeping the number of Q-levels fixed to N, so as to minimize the mean-square error E from equation (3.1). Note that the problem has the following constraints.

- For a given pair $(x, f(x))$, x is known only to A and the desired outcome $f(x)$ is known only to S. Hence gradient-based adaptation of the quantization map q cannot be done.

- Since we consider bit-based complexity measure of the messages exchanged between agent A and supervisor S (section 3.2), S can feedback to A only discretized information about the error between the desired outcome $f(x)$ and the response of the organization $f(q(x))$.
- Probability distribution P is, in general, not known beforehand. Thus we use the following empirical estimate of the error function instead:

$$\bar{E} = \frac{1}{m} \sum_{k=1}^{m} (\| f(x(k)) - f(q(x(k))) \|_2)^2. \tag{4.1}$$

In section 2.1, we discussed the competitive learning (CL) algorithm for VQ. We adapt the CL algorithm as our starting point and indicate how it can be modified to tackle the present problem.

As observed before, supervisor S can only give to A discretized information about the error between the desired outcome $f(x)$ and the response of the organization $f(q(x),)$. In particular, let us consider the case in which S can feedback to A only one bit of information about the error. For this purpose, assume that the goal of adaptation is to reduce the mean-square error E below a prespecified $\epsilon > 0$, that is,

$$E \equiv \int_D (\| f(x) - f(q(x)) \|_2)^2 \, P(dx) < \epsilon. \tag{4.2}$$

Now we let S feedback to A the following *reinforcement signal* r:

$$r(k) = \begin{cases} 1 & \text{if } \left(\| f(x(k)) - f(q(x(k))) \|_2\right)^2 < \epsilon, \\ 0 & \text{otherwise.} \end{cases}$$

On receiving r, agent A adapts its Q-levels y_i as follows:

$$y_i(k+1) = y_i(k) + \eta(k) \cdot (x(k) - y_i(k)) \cdot z_i(k) \cdot (1 - r(k)) \tag{4.3}$$

where $\eta(k)$ is the learning rate, and $z_i(k)$ is as in equation (2.3). In other words, agent A moves the winning Q-level towards the input only if the error exceeds a prespecified value. A particularly interesting choice of $\eta(k)$ is when it varies with different rates for different Q-levels $y_i, i = 1, \ldots, N$, as follows:

$$\eta_i(k) = \frac{1}{\sum_{j=1}^{k} z_i(j) \cdot (1 - r(j)) + 1} \tag{4.4}$$

which along with equation (4.3) gives

$$y_i(k+1) \approx \frac{1}{\sum_{j=1}^{k} z_i(j) \cdot (1 - r(j)) + 1} \cdot \left(\sum_{j=1}^{k} x(j) \cdot z_i(j) \cdot (1 - r(j)) + y_i(0) \right). \tag{4.5}$$

That is, equation (4.3) adapts the quantization map q such that each Q-level y_i goes to the *centroid* of all x for which it is at the minimum distance and representing x by y_i causes the error to exceed ϵ. Note that $y_i(k+1)$ is only approximately given by equation (4.5) because, in the adaptation rule equation (4.3), y_i is updated after every input x seen and, therefore, the region for which y_i is at the minimum distance, is changing.

To get an intuitive feel of the rule, consider the following example: Let $D = [0, 1]$, $N = 1$, $y_1(0) = 0.5$, $\epsilon = 0.1$, P is uniform over D, and the function f is as shown in Figure 2. We would like to adapt y_1 so as to minimize the error E. Now the adaptation rule equation (4.3) will give y_1 an expected *push* towards the right as that side of y_1 contributes more error than the other side. But that is what the adaptation rule is required to do in the single Q-level case.

Next, consider the same example but with $N = 2$ (i.e., the quantization map q has two Q-levels). Let $y_1(0) = 0.3$ and $y_2(0) = 0.8$. Now the expected push for both the Q-levels is 0. That is, since the two sides of $y_i, i = 1, 2$ contribute the same amount of error, they receive the same expected reinforcement signal r and, therefore, the expected change in y_i is 0. But the two Q-levels y_1 and y_2 contribute unequally to the error E (in fact, error occurs only in the second Q-level y_2). Such a problem occurs because the adaptation rule equation (4.3) updates y_i only to minimize its own error, and not the overall error. To overcome this problem, one way is to modify the metric d while computing z_is with equation (2.3) so as to ensure that, during the course of the training process, all Q-levels contribute approximately the same error (e.g., *conscience mechanism* [2], or *frequency-sensitive competitive learning* [1], can be suitably adapted for the task). Alternatively, we can adapt the number of Q-levels as discussed next.

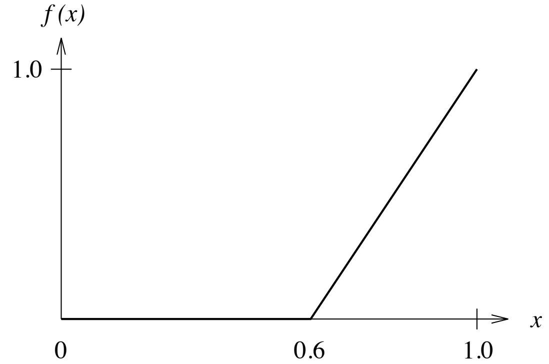


Figure 2. An example for the adaptation rule.

4.1.1 Adaptation of the number of quantization levels

This step of the algorithm helps not only in overcoming the limitations of the adaptation rule, but also in avoiding the requirement of a prespecified number of Q-levels. Moreover, this step becomes essential when there is more than one agent in the problem. But let us first consider the single-agent case.

We start by assigning two Q-levels to A (i.e., $N = 2$). We next adapt these Q-levels using the adaptation rule equation (4.3) for a fixed number, say γ, of input-output pairs. Also, we collect the error statistics for each Q-level; that is, we keep count of the number of times the ith Q-level has received the reinforcement signal r as 0, denoted by $\tilde{E}_i$. Then the value of $\tilde{E}_i$, after k input-output pairs have been seen, is

$$\tilde{E}_i(k) = \sum_{j=1}^{k} z_i(j) \cdot (1 - r(j))$$

where z_i is as in equation (4.3).

After the completion of γ adaptation steps, consider an estimate of the mean-square error $\tilde{E} = \sum_{i=1}^{N} \tilde{E}_i$. If $\tilde{E} < \epsilon$, then we are done as the aim of equation (4.2) has been achieved. Otherwise, we add a Q-level y_{N+1} between the maximum-error Q-level and its maximum-error neighbor. Specifically, let $i^* = \arg\max_{1\le i\le N} \tilde{E}_i$. Then add a Q-level between Q-level i^* and its maximum-error *neighbor*. Two Q-levels are said to be neighbors if their regions have a common boundary, that is, let σ_i be the region associated with ith Q-level, y_i (i.e., $\sigma_i = \{x : \arg\min_j \| x - y_j \|_2 = i\}$), then y_i and y_j are neighbors if $\bar{\sigma}_i \cap \bar{\sigma}_j \neq \emptyset$, where $\bar{\sigma}_i$ refers to the closure of the set σ_i). Let $NE(i)$ be the set of all Q-levels which are neighbors of i, and let $\beta(i) = \arg\max_{k\in NE(i^*)} \tilde{E}_k$. Then a Q-level is added between the maximum-error Q-level i^* and its maximum-error neighbor $\beta(i^*)$. The newly added Q-level y_{N+1} is initialized as:

$$y_{N+1} = \frac{y_{i^*} \cdot \tilde{E}_{i^*} + y_j \cdot \tilde{E}_{\beta(i^*)}}{\tilde{E}_{i^*} + \tilde{E}_{\beta(i^*)}}.$$

This procedure requires knowledge of the neighborhood set of each Q-level. A number of techniques exist to iteratively compute the neighborhood sets ([3, 12], and references therein). However, all of them are computation-intensive and impose specific restrictions on the topology of the Q-levels. We, therefore, use a crude-but-simple procedure: split the maximum-error Q-level i^* into two Q-levels i^*_{new} and $(N + 1)$. Let $j = \arg\max_{k\neq i^*} \tilde{E}_k$, that is, j is the maximum-error-but-one Q-level. Then the new Q-levels are computed as

$$\begin{aligned} y_{i^*_{\text{new}}} &= y_{i^*} + \Delta \cdot (y_j - y_{i^*}), \quad \text{and} \\ y_{N+1} &= y_{i^*} - \Delta \cdot (y_j - y_{i^*}) \end{aligned} \tag{4.6}$$

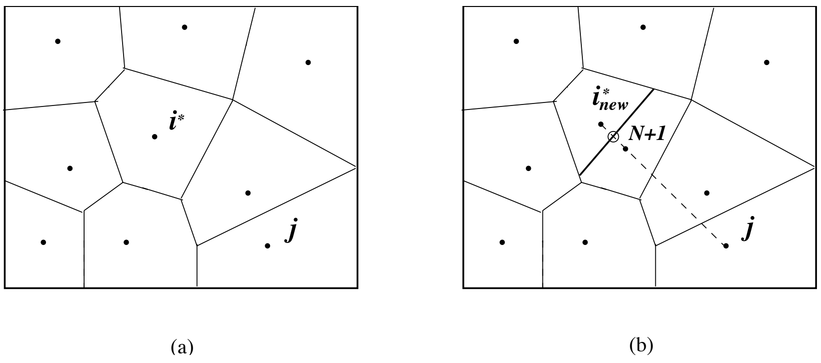


Figure 3. An illustration of the splitting procedure. (a) Quantization before splitting the maximum-error Q-level i^*, j is the maximum-error-but-one Q-level. (b) After splitting i^* into i^*_{new} and $N + 1$.

where Δ is a positive real number such that the set $\{x : \| x - y_{i^*} \|_2 \leq \Delta\}$ is contained in σ_{i^*}. In other words, the procedure splits the region σ_{i^*} into two regions with a hyperplane which passes through y_{i^*} and is orthogonal to the line joining y_{i^*} and y_j. Moreover, if Δ is small enough, then the other cells of the partition are not affected much by the split. Figure 3 illustrates the procedure for a two-dimensional case.

In addition to splitting the maximum-error Q-level, we delete a very-low-error Q-level. In particular, let $i_m = \arg\min_{1\leq i\leq N} \tilde{E}_i$ and $\tilde{E} = \sum_{i=1}^{N} E_i$. If $\tilde{E}_{i_m}/\tilde{E} < 0.1/N$, then we delete the Q-level i_m, that is, a Q-level contributing less than one percent of its share of $\tilde{E}$ is removed. Deletion is required because the adaptation rule, as well as the splitting procedure, is not exact. Hence there may exist a Q-level which has a small error compared to the rest of the Q-levels.

4.1.2 Simulation results for single-agent case

We now present simulation results for the adaptation procedure for DGVQ in the single-agent case on some synthetic problems. More specifically, we consider the two-level hierarchy where a single agent A reports to supervisor S. That is, when there is no decentralization, but just goal-based quantization.

Recall that the algorithm has four main parameters: the number of adaptation steps γ, the error threshold ϵ, the weight factor λ, and the stopping parameter $\Delta\epsilon$. The number of adaptation steps γ is chosen such that the error estimates obtained in those steps are *close* to the true values. Hence this parameter does not affect the simulations substantially as long as it is sufficiently large. In the simulation results given here we chose $\gamma = 200$. On the other hand, ϵ and λ are critical parameters, and are chosen carefully. ϵ is chosen depending on the accuracy to which

ϵ	λ	$\Delta\epsilon$	α	Q-levels(0)	γ	$\max_{\text{Latency}}$
0.01	0.01	0.001	0.7	1	200	$3 \cdot N$

Table 1. Default values for the parameters of the algorithm.

the quantized output is required to match the desired outcome function f, while λ decides the relative weight given to the bit complexity of the quantization maps as compared to the quantization error. A very small value of λ leads to a relatively large number of Q-levels. On the other hand, a large λ results in a small number of Q-levels at the cost of large error. Finally, the stopping parameter $\Delta\epsilon$ decides the minimum change in the energy function $\mathcal{H}$ required to continue with the adaptation of the number of Q-levels.

One point to note is that as the proposed algorithm is designed to work in real time, it is assumed that the training data is being continuously generated. Hence the problem of overfitting does not arise. However, if the algorithm is to be used in batch-mode (i.e., the algorithm is required to design DGVQ on a given set of data), the problem of overfitting the training data can be addressed through *complexity regularization*.

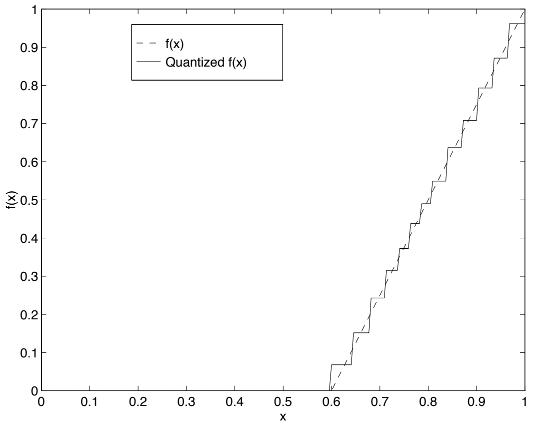
We first consider the case when A observes just a single one-dimensional variable x and $D = [0, 1]$. The probability distribution P is taken to be the uniform distribution over D. Let the desired outcome function f be as shown in Figure 2. The parameters of the algorithm take their values from Table 1. Figure 4 gives the output of the algorithm, as well as the corresponding energy *versus* number of iterations plot. Note that, as desired, the algorithm has created a single Q-level for $[0, 0.6]$, and has (approximately) uniformly quantized the interval $[0.6, 1.0]$. Moreover, the energy $\mathcal{H}$ decreases monotonically with the number of iterations.

We next summarize the simulation results for a few other desired outcome functions in Table 2. The corresponding graphical outputs are given in Figure 5.

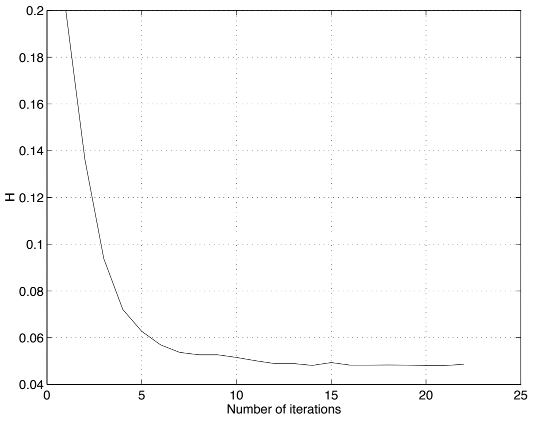
Now consider the case when agent A observes more than one variable. In particular, we assume that A observes two one-dimensional variables x and y, each lying in $[0, 1]$. Note that the problem still does not involve decentralization. Table 3 gives the performance of the algorithm on various $f(x, y)$. The parameter values are the same as before except that $\gamma = 500$, $\lambda = 0.001$, and $\Delta\epsilon = 0.0001$. Figure 6 gives the graphical output for $f(x, y) = x^y$.

4.2 Adaptation rule for multiagent case

Until now we have considered the single-agent case. We next generalize the procedure for the case when n agents report to supervisor S. The adaptation step remains the same, with all agents receiving the same



(a)

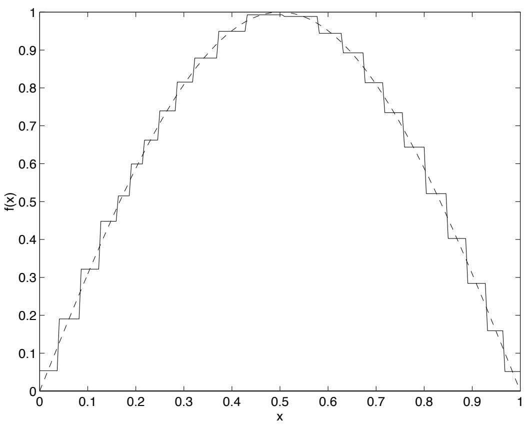


(b)

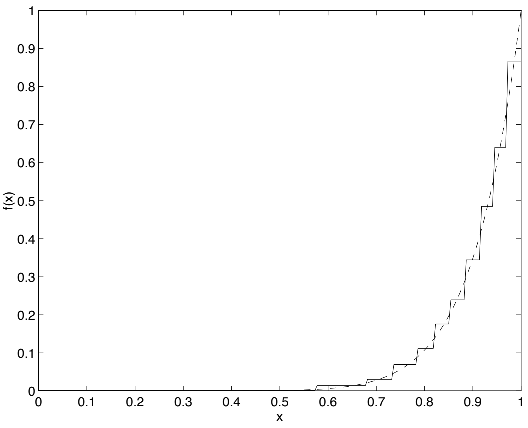
Figure 4. Output of the algorithm for $f(x)$ from Figure (2): (a) $f(x)$ and quantized $f(x)$ *versus* x, and (b) energy *versus* number of iterations.

Simu-lation	Desired outcome function $f(x)$	Iter-ations	Q-levels	Initial Error	Final Error	Final Energy
1	0, if $x < 0.6$; $(x - .6)/0.4$, otherwise	22	14	0.20	0.0082	0.086
2	$\sin(\pi \cdot x)$	23	23	0.37	0.0210	0.072
3	x^{10}	15	11	0.09	0.0092	0.047
4	$50(x - .1)(x - .2)$ $\times(x - .5)(x - .8)$	23	23	0.36	0.0380	0.094
5	$2 \cdot x$, if $x < 0.6$; $(1 - x)$, otherwise	32	20	0.60	0.0240	0.075

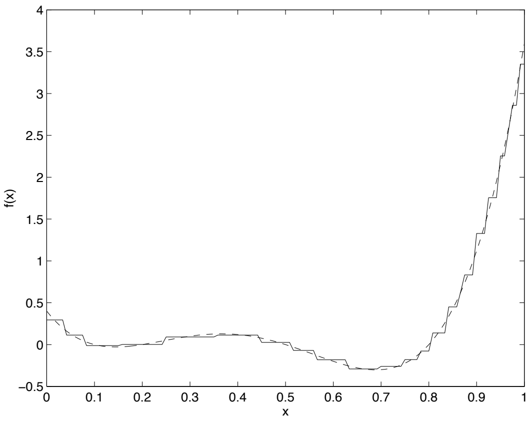
Table 2. Simulation results for various functions when agent A observes a single variable x.



(a)

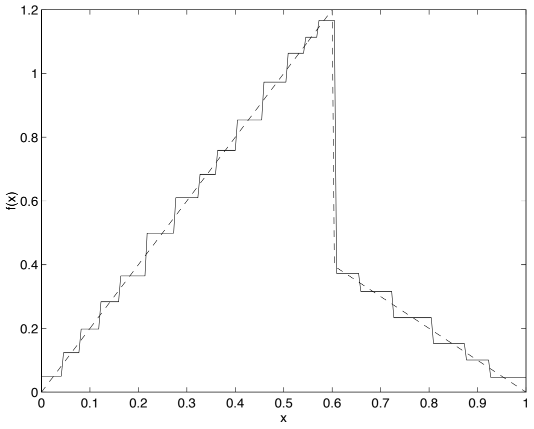


(b)



(c)

Figure 5. Output of the algorithm for various desired outcome functions: (a) $f(x) = \sin(\pi \cdot x)$, (b) $f(x) = x^{10}$, (c) $f(x) = 50(x - 0.1)(x - 0.2)(x - 0.5)(x - 0.8)$, and (d) $f(x) = 2 \cdot x$, if $x < 0.6$; $(1 - x)$, otherwise. Refer to Table 2 for other details.



(d)

Figure 5(continued).

Simu-lation	Desired outcome function $f(x,y)$	Iter-ations	Q-levels	Initial Error	Final Error	Final Energy
1	$2 \cdot \sin(\pi \cdot x)$ $+1 \cdot \sin(\pi \cdot y)$	80	70	1.08	0.144	0.187
2	$x^2 \cdot y$	74	56	0.15	0.021	0.032
3	x^y	90	66	0.22	0.035	0.047
4	$e^{-10 \cdot [(x-.25)^2 + (y-.75)^2 - (x-.25) \cdot (y-.75)]}$	42	38	0.16	0.038	0.055

Table 3. Simulation results for various functions when agent A observes two one-dimensional variables x and y.

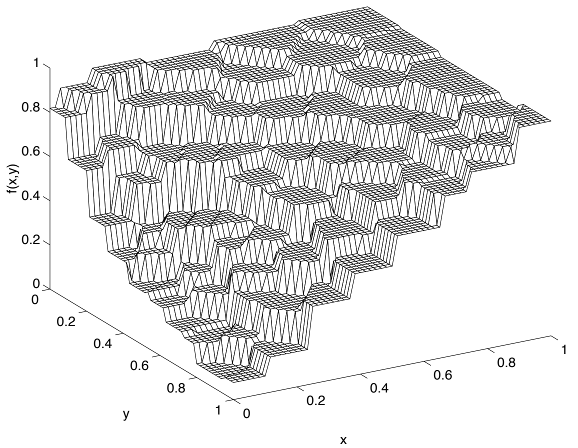


Figure 6. The case where agent A observes two variables x and y: $f(x,y) = x^y$.

reinforcement signal r. That is, each agent adapts its quantization map using the adaptation rule equation (4.3) for γ number of input-output pairs. After γ adaptation steps, each agent updates its number of Q-levels as if it were the only agent reporting to S, that is, each agent adds (deletes) Q-levels as discussed for the single-agent case.

The problem with this simple procedure is that the agents end up having approximately the same number of Q-levels irrespective of the actual dependence of the outcome function on their local environments. Such a problem occurs because all the agents receive the same reinforcement signal r, and the agents adapt their number of Q-levels independently of each other. The former is a constraint arising due to decentralization and cannot be avoided unless supervisor S has explicit knowledge of the relative dependence of the outcome function on each agent's environment. But the latter can be circumvented as discussed next.

4.2.1 Differential adaptation of the number of Q-levels

As indicated previously, we need a way to ensure that the agents have a number of Q-levels in proportion to the relative dependence of the outcome function on their local environment. That is, if the outcome function f has less variation with respect to local environment x_i of agent A_i than with respect to x_j, the number of Q-levels for A_i, N_i, should be proportionately less than N_j. We now discuss two methods for achieving the goal: credit assignment and sequential adaptation.

Credit assignment method

The basic idea has been adapted from the *reinforcement learning* literature (a good survey can be found in [13]). In the problems considered, the reinforcement signal r for an action arrives after a random amount of time since the action has been taken. Hence credit for receiving r is assigned among all the actions taken until the arrival of r. Such an assignment problem is usually referred to as *temporal credit assignment*. In our problem, all agents receive the same error signal $(1 - r)$. We call the problem of assigning credit for the error signal among the agents *spatial* or *structural credit assignment*.

More precisely, let c_i denote the credit assigned to agent A_i for the present error probability, $\bar{E} = P(\{r = 0\})$, where $0 \le c_i \le 1$ and $\sum_{i=1}^{n} c_i = 1$. The procedure for adapting the number of Q-levels is as follows. As before, each agent A_i starts with two Q-levels and $c_i = 1/n$. Agents then adapt their quantization maps for γ input-output pairs using the adaptation rule equation (4.3). After the completion of γ adaptation steps, each agent A_i adapts its number of Q-levels N_i so as to minimize the following function:

$$\bar{\mathcal{H}}_i = c_i \cdot P(\{r = 0\}) + \lambda \cdot \log_2 N_i. \tag{4.7}$$

Hence c_i *decentralizes* the process of minimizing the original energy function $\mathcal{H}$ from equation (3.2). Note that equation (4.7) uses the probability of error exceeding ϵ instead of the mean-square error because the agents receive information from the supervisor only about the former and not the latter.

Agent A_i adapts N_i as follows: let w_i be 1 (resp. -1) if agent A_i added (resp. deleted) a Q-level in the previous iteration. One iteration refers to the completion of γ adaptation steps. Also, let $\Delta\bar{\mathcal{H}}_i$ indicate the change in $\bar{\mathcal{H}}_i$ during the previous iteration. Then

$$w_i \ := \ \operatorname{sgn}\left(-w_i \cdot \Delta\bar{\mathcal{H}}_i\right)$$

where $\operatorname{sgn}(x) = 1$ if $x \geq 0$ else $\operatorname{sgn}(x) = -1$. In other words, A_i continues to take the action (addition or deletion of a Q-level) taken in the previous adaptation if the action resulted in a decrease in $\bar{\mathcal{H}}_i$, otherwise it takes the opposite action.

Now observe that c_i, $i = 1, \ldots, n$, are themselves dependent on N_i. Hence c_i is also adapted with time, but at a slower rate compared to N_i. In particular, c_i can be adapted after a fixed number, say α, of N_i adaptations. Call the duration between two adaptations of c_i an *epoch*. Let $\tilde{\mathcal{H}} = 1/\alpha \sum_{k=1}^{\alpha} \bar{\mathcal{H}}(k)$, that is, $\tilde{\mathcal{H}}$ indicates the average value of the energy function over an epoch. Then the adaptation rule for c_i is:

$$c_i \ := \ c_i + \eta \cdot \operatorname{sgn}\left(-\Delta\tilde{\mathcal{H}} \cdot \Delta c_i\right) \tag{4.8}$$

where η is the learning rate, and $\Delta\tilde{\mathcal{H}}$ and Δc_i indicate the respective changes in $\tilde{\mathcal{H}}$ and c_i over the previous epoch. After every adaptation of c_i, project and normalize c_i so as to ensure that $0 \leq c_i \leq 1$ and $\sum_{i=1}^{n} c_i = 1$.

Sequential adaptation method

Though the previous method adapts the number of Q-levels simultaneously for all agents, it requires an additional loop for updating the credits c_i. Moreover, the performance of the method in simulations is not very satisfactory. Hence we now describe a procedure which adapts the number of Q-levels for only one agent at a time. As in equation (3.2), let the goal of adaptation be to minimize the following energy function:

$$\mathcal{H} = \int_D (\| f(x) - f(q_1(x_1), \ldots, q_n(x_n)) \|_2)^2 P(dx) + \lambda \cdot \sum_{i=1}^{n} \log_2 N_i. \tag{4.9}$$

Now the basic idea is to let supervisor S decide which agent is to adapt its number of Q-levels in the present iteration. For this purpose, S keeps track of the change in the energy function $\mathcal{H}$, $\Delta\mathcal{H}_i$, resulting from the previous adaptation of the number of Q-levels of agent A_i (i.e., N_i). S then asks the agent that induced maximum change in $\mathcal{H}$ in its previous adaptation, to update its number of Q-levels. Specifically, let $\beta(k-1)$ be the agent that adapted its number of Q-levels after the $(k-1)$th iteration (recall that one iteration refers to the completion of γ adaptation steps). Also, let

$$w_i(k-1) = \begin{cases} +1 & \text{if } i = \beta(k-1) \text{ and} \\ & \beta(k-1) \text{ added a Q-level,} \\ -1 & \text{if } i = \beta(k-1) \text{ and} \\ & \beta(k-1) \text{ deleted a Q-level,} \\ w_i(k-2) & \text{otherwise.} \end{cases}$$

That is, $w_i = 1$ (resp. -1) indicates that agent A_i added (resp. deleted) a Q-level in its previous adaptation of the number of Q-levels. Next, let

$$\Delta\mathcal{H}_i(k) = \begin{cases} \mathcal{H}(k) - \mathcal{H}(k-1) & \text{if } i = \beta(k-1), \\ \Delta\mathcal{H}_i(k-1) & \text{otherwise.} \end{cases} \tag{4.10}$$

Then, after the kth iteration, S asks the $\beta(k)$th agent to update its number of Q-levels according to $w_{\beta(k)}(k)$, where

$$\begin{aligned} \beta(k) &= \arg\max_{1\le i\le N} |\Delta\mathcal{H}_i(k)| \\ w_{\beta(k)}(k) &= \operatorname{sgn}(-w_{\beta(k)}(k-1)\cdot\Delta\mathcal{H}_{\beta(k)}(k)). \end{aligned} \tag{4.11}$$

Equivalently, S asks the $\beta(k)$th agent to add a Q-level if either its previous adaptation was the addition of a Q-level that resulted in a reduction of $\mathcal{H}$, or its previous adaptation was the deletion of a Q-level that increased $\mathcal{H}$. Otherwise, S asks $\beta(k)$ to delete a Q-level.

This procedure is repeated until $\max_i |\Delta\mathcal{H}_i(k)| < \Delta\epsilon$, where $\Delta\epsilon > 0$ is a prespecified value. In other words, the training process is stopped if none of the agents are able to substantially reduce the energy function by adapting its number of Q-levels.

For this procedure we assumed that we had an exact estimate of $\mathcal{H}(k)$, an optimal adaptation rule, and an optimal splitting procedure. Since that is not the case, we further modify the procedure as follows.

- Since $\mathcal{H}(k)$ is estimated from a finite number of samples, it will have some *variance* around a mean value. To reduce the effect of this variance, we average $\Delta\mathcal{H}_i(k)$ as follows:

  $$\overline{\Delta\mathcal{H}_i}(k) = \begin{cases} \alpha\cdot\overline{\Delta\mathcal{H}_i}(k-1) + (1-\alpha)\cdot\big(\mathcal{H}(k) - \mathcal{H}(k-1)\big) & \text{if } i = \beta(k-1), \\ \overline{\Delta\mathcal{H}_i}(k-1) & \text{otherwise} \end{cases}$$

  where $\alpha \in [0,1]$.

- In general, $\overline{\Delta\mathcal{H}_i}$ is dependent not only on N_i, but also on the number of Q-levels in the quantization maps of the rest of the agents. For example, consider the case where two agents, each observing one variable x and y, report to supervisor S. Let the desired outcome function be $f(x,y) = x^y$. Then the change in the energy function due to adaptation of the number of Q-levels of x is also dependent on the current number of Q-levels for

y. Now, in one iteration, only one agent updates its $\overline{\Delta\mathcal{H}_i}$. Hence it is possible that an agent last adapted its number of Q-levels (and thus its $\overline{\Delta\mathcal{H}_i}$) a large number of iterations ago. In such a case, that agent's $\overline{\Delta\mathcal{H}_i}$ may not be a true reflection of the agent's *potential* in reducing the energy function.

Therefore, if an agent, say i_0, has not updated its $\overline{\Delta\mathcal{H}_{i_0}}$ for the last $\max_{\text{Latency}}$ number of iterations, then $\beta(k)$ is set to i_0 and $w_{i_0}(k) = 1$ (i.e, i_0 adds a Q-level, so as to update its $\overline{\Delta\mathcal{H}_{i_0}}$). $\max_{\text{Latency}}$ is taken to be a multiple of the number of agents in the organization.

- Finally, since the adaptation rule, as well as the splitting procedure, is nonoptimal, each agent periodically deletes a very-low-error Q-level. This deletion is separate from the deletion due to equation (4.11).

Remark 4.1. In this algorithm, we have assumed that supervisor S can feedback to agents only a single bit of information about the quantization error E; that is, the reinforcement signal r can take only two values, 0 or 1. The algorithm can be easily extended to the more general case: when S can feedback l bits of information. The supervisor now transmits to agents an l-bit reinforcement signal, $r \equiv (r_l, \ldots, r_1) \in \{0,1\}^l$, which encodes the quantization error E such that a higher value of r indicates lower E. Let $\mathcal{R} : \{0,1\}^l \to [0,1]$, be defined by $\mathcal{R}(r) = 1/2^l \sum_{i=1}^{l} 2^{i-1} \cdot r_i$. Then the agents adapt their quantization maps using the adaptation rule equation (4.3), with r being replaced by $\mathcal{R}(r)$. The rest of the algorithm remains the same.

Remark 4.2. While arriving at this algorithm, we have assumed that supervisor S has knowledge of the quantization map q_i of agent A_i, for each i. But A_i adapts q_i after every input-output pair seen by the organization, and therefore, A_i needs to transmit the changes in q_i to S. For this purpose we can discretize the adaptation rule, that is, q_i is allowed to take values only over a prespecified grid on D_i. Then, after every adaptation of q_i, A_i transmits the number of steps by which q_i has moved over the grid. Or, to reduce the communication overhead, A_i can transmit to S the net change in its quantization map q_i after the completion of each iteration (i.e., γ adaptation steps), and not after every adaptation step.

4.2.2 Simulation results for multiagent case

We now present simulation results for the given procedure for DGVQ in the multiagent case. We first discuss the case when two agents, A_1 and A_2, report to supervisor S. Moreover, for the sake of simplicity, we assume that A_1 and A_2 observe only one variable each: x and y respectively.

We first consider the desired outcome function $f(x,y)$ to be such that decentralization of the quantization process is "natural," that is, $f(x,y)$

Simulation	A	B	Iterations	Q-levels for x	Q-levels for y	Initial Error	Final Error	Final Energy
1	2.0	2.0	50	23	22	1.45	0.063	0.35
2	1.0	2.0	40	14	23	1.10	0.056	0.32
3	0.5	2.0	40	8	23	0.92	0.056	0.30
4	0.0	2.0	36	1	20	0.74	0.050	0.20
5	2.0	0.0	33	23	1	0.71	0.042	0.21
6	2.0	0.5	35	20	8	0.90	0.060	0.30
7	2.0	1.0	39	22	14	1.08	0.059	0.32

Table 4. Simulation results for the two-agent case. The agents observe one variable each, x and y respectively. The desired outcome function is $f(x,y) = A \cdot \sin(\pi \cdot x) + B \cdot \sin(\pi \cdot y)$.

Simulation	Desired outcome function $f(x,y)$	Iterations	Q for x	Q for y	Initial Error	Final Error	Final Energy
1	$x^2 \cdot y$	37	16	14	0.15	0.011	0.092
2	x^y	34	17	12	0.22	0.018	0.099
3	$e^{-10[(x-.25)^2+(y-.75)^2-(x-.25)(y-.75)]}$	42	14	19	0.16	0.015	0.059
4	$4\sin(\pi \cdot x \cdot y)$	41	17	21	1.29	0.088	0.370

Table 5. Simulation results for various $f(x,y)$ for the two-agent case.

can be written as $g(x) + h(y)$. In particular, we consider

$$f(x,y) = A \cdot \sin(\pi \cdot x) + B \cdot \sin(\pi \cdot y) \tag{4.12}$$

where $A, B \in \mathbb{R}$. Table 4 gives the outputs of the algorithm for various values of A and B. The parameter values are taken from Table 1. Note that, as the result of using *differential adaptation of Q-levels* (section 4.2.1), the algorithm creates the number of Q-levels for the two agents in proportion to the relative dependence of the outcome function on their local environments.

Now consider $f(x,y)$ for which decentralization is not natural. Table 5 gives the output of the algorithm for various $f(x,y)$. Figure 7 gives the graphical output for $f(x,y) = x^y$. Comparing with the corresponding entries in Table 3, we expectedly find that the latter performs better than the former.

We now consider the case when more than two agents report to supervisor S. This will illustrate the performance of the algorithm for higher-dimensional input. Here we discuss two examples.

In the first example, four agents $A_i, 1 \le i \le 4$, each observing one variable x_i, report to S. The desired outcome function is taken to be:

$$f(x_1,x_2,x_3,x_4) = x_1 \cdot x_2^2 + x_1 \cdot x_3 \cdot x_4. \tag{4.13}$$

The second example also has four agents, but now A_1 and A_4 observe two variables each: $x_{ij}, j = 1,2;\ i = 1,4$. A_2 and A_3 continue to observe

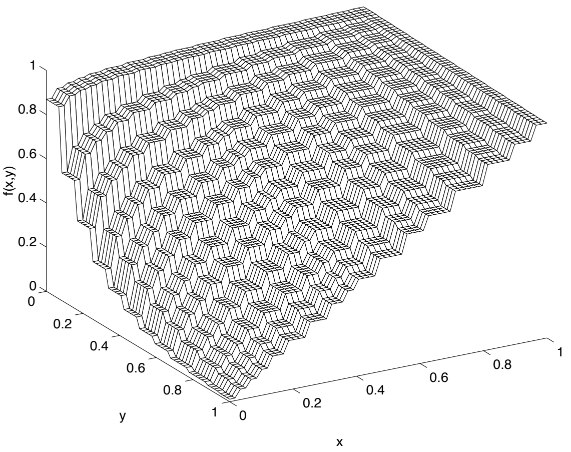


Figure 7. Two-agent case, each agent observes one variable (x and y resp.), $f(x, y) = x^y$.

Simu-lation	$f(\cdot)$	Iter-ations	Q for A_1	Q for A_2	Q for A_3	Q for A_4	Initial Error	Final Error	Final Energy
1	(4.13)	88	27	22	17	13	0.21	0.014	0.032
2	(4.14)	285	46	35	25	79	0.64	0.120	0.340

Table 6. Two examples for the four-agent case.

one variable each. The desired outcome function is:

$$f(\mathbf{x}) = x_{11} \cdot \sin(\pi \cdot x_2) + 2 \cdot x_{12} \cdot x_3^2 \cdot x_{41} + e^{-5(x_{42}-0.25)(x_3-0.6)}. \qquad (4.14)$$

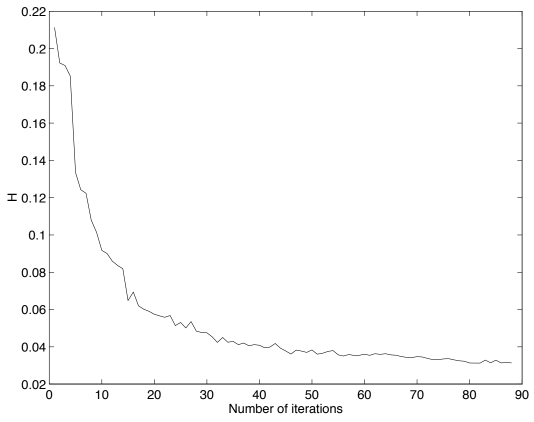
Table 6 gives the performance of the algorithm on the two examples. The parameter values are as in Table 1 except that $\lambda = 0.001$ and $\Delta\epsilon = 0.0001$, and in the second example $\gamma = 500$. Figure 8 gives the corresponding energy *versus* number of iterations plots.

5. Extensions

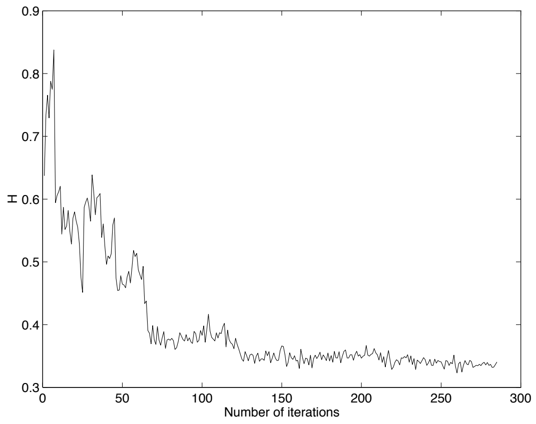
In this section we discuss the following extensions of the DGVQ problem: unknown desired outcome function, multilevel hierarchy, and time-varying outcome function.

5.1 Unknown desired outcome function

In the preceding section, we discussed an algorithm for the two-level hierarchy example of section 3.2. We assumed that the desired outcome function f was known to supervisor S to the extent that given an input x, S could evaluate $f(x)$. We now relax this condition and indicate how our algorithm can be modified so as to learn the quantization maps as well as the unknown f.



(a)



(b)

Figure 8. Energy *versus* number of iterations plots for the four-agent case: (a) Example 1, and (b) Example 2.

The basic idea is to use a NN to learn the outcome function f. Specifically, let $\tilde{f}$ be the approximation of f learned by the organization, then the goal is to minimize the following modified energy function:

$$\mathcal{H} = \int_D \left(\| f(x) - \tilde{f}\,(q_1(x_1), \ldots, q_n(x_n)) \|_2 \right)^2 \, P(dx) + \lambda \cdot \sum_{i=1}^{n} \log_2 N_i.$$

Supervisor S uses a feedforward NN $\mathcal{N}$ to learn f, with input to $\mathcal{N}$ being $\hat{x} \equiv (q_1(x_1), \ldots, q_n(x_n))$ and the desired output being $f(x)$. The NN $\mathcal{N}$ is trained using the *backpropagation* algorithm so as to minimize the mean-square error, $E[(\| f(x) - \tilde{f}(\hat{x}) \|_2)^2]$. Simultaneously, the agents $A_i, i = 1, \ldots, n$ learn their quantization maps q_i using the algorithm of

Simulation	Desired outcome function $f(x)$	Iterations	Q-levels	Initial Error	Final Error	Final Energy
1	$0.6 \cdot x^{10} + 0.2$	20	10	0.09	0.01	0.04
2	$0.6 \cdot \sin(\pi \cdot x) + 0.2$	38	17	0.20	0.02	0.06
3	$0.5 \sin(\pi \cdot x)$ $+0.2 \sin(\pi \cdot y) + 0.15$	103	71	0.23	0.04	0.10
4	$0.7 \cdot x^2 \cdot y + 0.2$	44	41	0.14	0.02	0.04

Table 7. Unknown desired outcome function, $f(x)$ for the single-agent case: 1 and 2 for when A observes one variable x, and 3 and 4 for when A observes two variables x and y.

Simulation	Desired outcome function $f(x,y)$	Iterations	Q for x	Q for y	Initial Error	Final Error	Final Energy
1	$0.5 \sin(\pi \cdot x)$ $+0.2 \sin(\pi \cdot y) + 0.15$	32	22	5	0.23	0.01	0.08
2	$0.7 \cdot x^2 \cdot y + 0.2$	44	11	9	0.14	0.02	0.07

Table 8. Various unknown desired outcome functions for the two-agent case.

the previous section. One modification to the algorithm is that each agent initializes its quantization maps with a sufficiently large number of Q-levels, instead of just one Q-level. This is required in order to stabilize the simultaneous learning of the unknown outcome function and the quantization maps.

5.1.1 Simulation Results

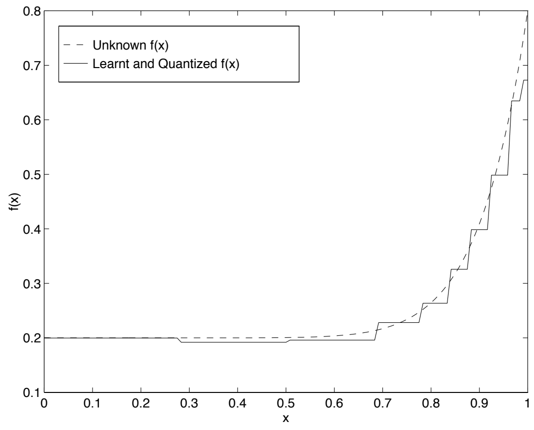
We now present simulation results for the preceding case, that is, when the desired outcome function f is unknown. In all the examples considered, we have used a two-layer feedforward NN with 10 and 1 sigmoidal neurons in the first and the second layers respectively. The parameters are taken from Table 1 except that $\gamma = 500$ and the starting number of Q-levels is 3. Larger values are chosen because now the desired outcome function is also being learned.

As before, we first discuss the case when a single agent A reports to supervisor S. Table 7 gives the output of the algorithm for some unknown desired outcome functions. Figure 9 gives some sample plots.

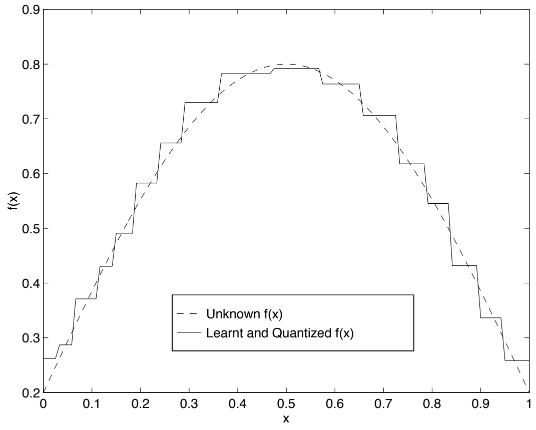
We then consider the case when two agents A_1 and A_2, observing x and y respectively, report to supervisor S and the desired outcome function $f(x,y)$ is unknown. Table 8 gives the output of the algorithm for two $f(x,y)$.

5.2 Multilevel hierarchy

Until now we have discussed the two-level hierarchy model of an organization. We next consider multilevel hierarchies.



(a)



(b)

Figure 9. Single-agent case with unknown desired outcome function, $f(x)$: (a) $f(x) = 0.6 \cdot x^{10} + 0.2$, (b) $f(x) = 0.6 * \sin(\pi \cdot x) + 0.2$.

In a multilevel hierarchy model, an organization has a *hierarchy* of agents and supervisors. A hierarchy is a connected and directed acyclic graph $\mathcal{G} \equiv (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ is the vertex (resp. agent) set of the graph (resp. organization) and $\mathcal{E}$ is the set of directed edges in the graph $\mathcal{G}$. A directed edge from vertex A_1 to vertex A_2 indicates that the corresponding members of the organization have an agent-supervisor relationship. Equivalently, we say that A_1 is a *subordinate* of A_2, and A_2 is a *superior* of A_1. Since $\mathcal{G}$ is acyclic, each member has well-defined superiors and subordinates; that is, it cannot happen that one is a subordinate as well as a superior (according to the given definitions) of another. Next we assume that the hierarchy can be organized in

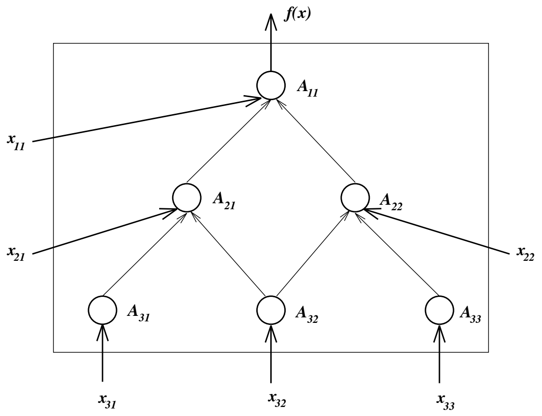


Figure 10. A three-level hierarchy model. A_{ij} is the jth agent in the ith level of the hierarchy.

levels such that each member in the hierarchy is a superior only of those belonging to one particular level, and each member is subordinate only to those of another particular level. Note that if we are allowed to introduce *dummy* agents—those that just output their inputs—then any hierarchy can be modified to satisfy the given requirement. With this assumption, one way of indexing the levels is: members that have no superior are assigned level 1, their subordinates are assigned level 2, and so on. Figure 10 illustrates a three-level hierarchy.

A multilevel hierarchy operates as follows: each member in the hierarchy observes their local environment as well as receives messages from their subordinates. Each member then evaluates its input-output function, and reports the outcomes to their superiors. Now the objective is to design the messages to be exchanged between the subordinate-superior pairs in the organization, so as to minimize a given energy function. As before, we assume that each member in the hierarchy knows its input-output function. Moreover, the energy function continues to be a weighted sum of the mean-square error and the bit-based information-exchange complexity of the hierarchy, that is,

$$\mathcal{H} = \int_D \left(\| f(x) - f(Q(x)) \|_2 \right)^2 P(dx) + \lambda \cdot \sum_{h=1}^{H} \sum_{i=1}^{n_h} \log_2 N_i^{(h)}$$

where H indicates the number of levels in the hierarchy, n_h the number of agents at the hth level, and $Q(x)$ denotes the overall effect on x due to quantization at various levels of the hierarchy.

Now such a hierarchy can be trained—each member in the hierarchy learns its quantization maps—in stages using the algorithm of the previous section. That is, first the agents at the highest level are trained to learn their quantization maps, then the agents in the next level learn, and so on. While agents at the hth level are being trained, those at higher levels use their learnt quantization maps before exchanging messages with their superiors. All the agents at levels below the hth level act *transparently*. That is, they do not perform any quantization before exchanging messages with their superiors.

Note that if all the external variables are observed at the highest level in the hierarchy, then, in the given procedure, the domain of the quantization map of each member at levels other than the highest level is discrete. Hence the training process for levels other than the highest level involves learning the quantization maps on a discrete domain, and is, therefore, fast and simple. However, learning at the highest level requires that members at all the other levels are able to transmit and receive real-valued variables. Since on-line learning requires that all members in the hierarchy exchange only bit-based messages, this procedure cannot be used for on-line learning. One way to overcome this problem is to let each member uniformly quantize the range of its input-output mapping with a large number of Q-levels. Then, while the agents at the highest level are being trained, each member at any other level uses that quantization map before transmitting the outcome to their superiors.

5.2.1 Simulation Results

We now give simulation results for learning the quantization maps in multilevel hierarchies. We assume that each member in the hierarchy knows its input-output function. In the following, we illustrate the procedure on three hierarchies.

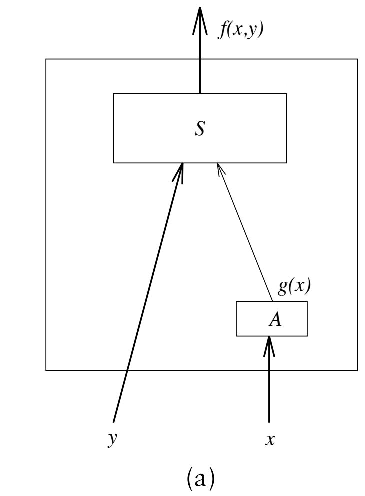
Example 1. The hierarchy is as shown in Figure 11(a). The hierarchy has a single subordinate-superior pair, and is similar to the single-agent single-supervisor case considered previously except for two differences. First, the agent A not only quantizes its environment x, but also evaluates its input-output map $g(\cdot)$ on the environment x and sends the quantized outcome to the supervisor. Second, S, in addition to receiving messages from A, observes its own local environment y. The desired outcome function is $f(x,y)$.

Here we take

$$\begin{aligned} g(x) &= \sin(\pi \cdot x) \\ f(x,y) &= y \cdot g(x). \end{aligned} \tag{5.1}$$

The performance of the algorithm is given in Table 9. Figures 12 and 13(a) give, respectively, the output and the energy *versus* iterations plot of the algorithm.

Example 2. The hierarchy is as shown in Figure 11(b). The hierarchy is similar to that of Example 1 except that now two agents A_1 and A_2, observing variables x and z respectively, report to supervisor S. The



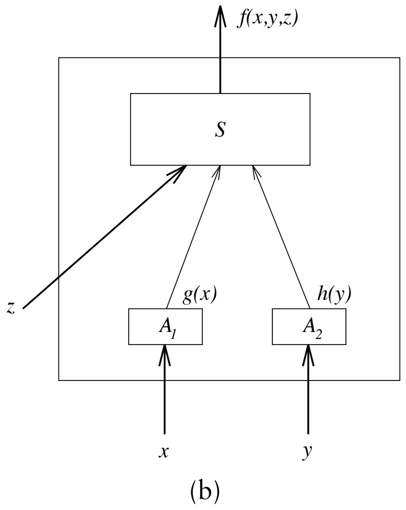


Figure 11. Multilevel hierarchies: (a) Example 1, and (b) Example 2.

Iterations		Q-levels			Initial	Final	Final
Stage I	Stage II	x	$g(x)$	y	Error	Error	Energy
26	38	24	12	13	0.23	0.019	0.15

Table 9. Multilevel hierarchy: Example 1.

Iterations		Q-levels					Initial	Final	Final
Stage I	Stage II	x	y	z	$g(x)$	$h(y)$	Error	Error	Energy
36	52	17	20	21	9	11	0.73	0.033	0.24

Table 10. Multilevel hierarchy: Example 2.

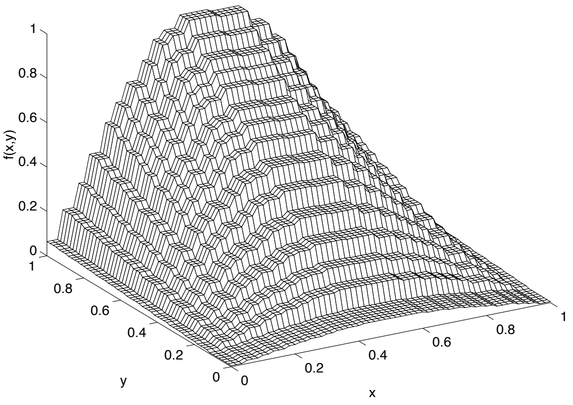
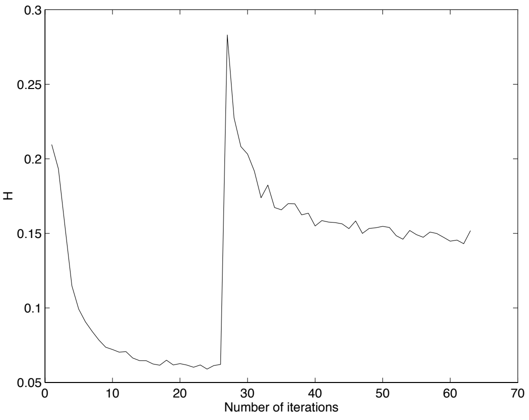
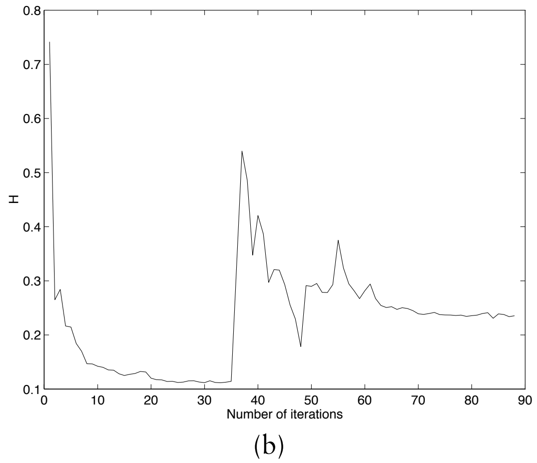


Figure 12. Multilevel hierarchy: Example 1.



(a)



(b)

Figure 13. Energy *versus* number of iterations: (a) Example 1, and (b) Example 2. The sudden jump in the energy value indicates the switch in the training of one stage to that of the next.

input-output functions of A_1 and A_2 are $g(\cdot)$ and $h(\cdot)$ respectively. As before, S receives the messages from A_1 and A_2 and also observes its own local environment y. The desired outcome function is $f(x, y, z)$. Here we take

$$\begin{aligned} g(x) &= \exp\left(-20 \cdot (x-0.5)^2\right) \\ h(y) &= \sin(\pi \cdot y) \\ f(x,y,z) &= g(x) \cdot h(y) + 2 \cdot (z-0.3)^3. \end{aligned} \tag{5.2}$$

The performance of the algorithm is given in Table 10. Figure 13(b) gives the energy *versus* iterations plot.

Example 3. Now we consider the three-level hierarchy of Figure 10. Let f_{ij} be the input-output map of the jth agent in the ith level of the hierarchy. In the present example, we choose

$$\begin{aligned} f_{31}(x_{31}) &= 2 \cdot \frac{\sin(5 \cdot \pi \cdot (x_{31}-0.5))}{5 \cdot \pi \cdot (x_{31}-0.5)} \\ f_{32}(x_{32}) &= 3 \cdot x_{32} \cdot (x_{32}-0.4) \cdot (x_{32}-0.7) \\ f_{33}(x_{33}) &= \exp\left(-5 \cdot (x_{33}-0.6)^2\right) \\ f_{21}(x_{21}, f_{31}, f_{32}) &= 2 \cdot x_{21} \cdot f_{31} + f_{32} \\ f_{22}(x_{22}, f_{32}, f_{33}) &= \frac{x_{22}+f_{32}+f_{33}}{2} \\ f_{11}(x_{11}, f_{21}, f_{22}) &= x_{11} \cdot f_{21} + f_{22}^2. \end{aligned} \tag{5.3}$$

The performance of the algorithm is given in Table 11. Figure 14 gives the energy *versus* iterations plot.

5.3 Time-varying outcome function

An important generalization of the DGVQ problem is when the desired outcome function itself varies with time. Such a model of an organization is clearly more realistic than when the outcome function is fixed.

Let the desired outcome function $f(t)$ vary "slowly" (clarified later) and continuously with time t. To learn $f(t)$ we use our algorithm (section 4) to learn the quantization maps of each agent, assuming that

Iterations			E_i	E_f	$\mathcal{H}$						
I 88	II 113	III 63	0.82	0.063	0.57						
Q-levels											
x_{31}	x_{32}	x_{33}	x_{21}	f_{31}	f_{32}	f_{32}	f_{33}	x_{22}	x_{11}	f_{21}	f_{22}
42	22	24	17	21	11	7	19	19	22	22	13

Table 11. Multilevel hierarchy: Example 3.

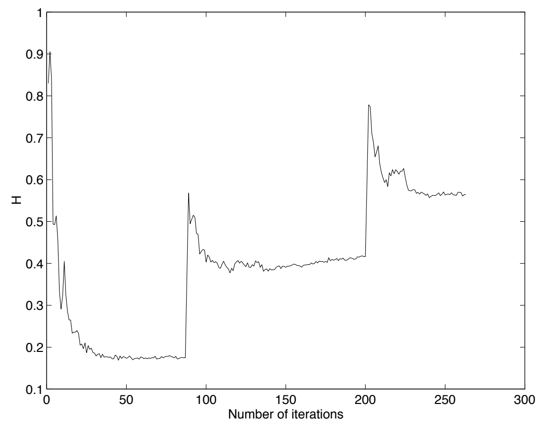


Figure 14. Example 3: Energy *versus* number of iterations.

$f(t)$ varies slowly enough as to be approximately constant over the time required to learn the quantization maps. After the completion of the learning process, supervisor S continues to observe the energy function $\mathcal{H}$ from equation (4.9). Once S realizes that $\mathcal{H}$ exceeds the value of the energy function at the completion of the learning process by a prespecified threshold, the learning process is initiated again. More precisely, let t_o be the time instant when the previous learning process was completed, with the value of the energy function being $\mathcal{H}(t_o)$. Once $(\mathcal{H}(t) - \mathcal{H}(t_o)) \geq \Delta$, where $\Delta > 0$ is a prespecified real number, the learning process is started all over again.

6. Conclusions

We have considered a generalization of classical vector quantization, which we have called decentralized goal-based vector quantization (DGVQ). The problem is to (vector) quantize or, equivalently, to partition the domain of a given function such that each cell in the partition satisfies a given set of topological constraints (e.g., each cell of the partition may be required to have the *cartesian-product* property). We have proposed an adaptive algorithm for DGVQ. The algorithm learns the quantization maps having the desired topological properties, by minimizing an appropriately defined *energy* function. We have indicated how a two-level hierarchy model of an economic organization can be formulated as DGVQ. We have also discussed a number of other generalizations: multilevel hierarchy model, DGVQ when the desired outcome function of the organization is unknown, and DGVQ when the function is time-varying. We have given extensive simulation results of the proposed algorithm on a number of synthetic problems.

Many aspects of the proposed learning scheme need further investigation as briefly explained in the following.

For training a multiagent two-level hierarchy, we have suggested two methods: namely, the credit assignment method and the sequential adaptation method. In the former, all the agents adapt their number of Q-levels simultaneously, but the method requires an additional loop to update the credits assigned to the agents for the quantization error. Because of this reason—and that the method did not perform very well in simulation—we have proposed the sequential adaptation method, in which only one agent adapts its number of Q-levels in one iteration. Now this method is inherently sequential, and takes a large number of iterations to converge for organizations having a large number of agents. Thus, it is desirable to have a method in which all agents simultaneously adapt their number of Q-levels. One way could be to reconsider the credit assignment method: the requirement of the additional loop for updating the credits can be avoided by adapting the credits after each iteration, and only those agents whose credits exceed the average credit value by a prespecified threshold, adapt their number of Q-levels.

For a multilevel hierarchy, we have suggested a method in which only one level of the hierarchy was trained at a time. In the resource allocation problem, it is more desirable that the agents, irrespective of the level to which they belong, learn their quantization maps simultaneously. For this purpose, the following generalization of the credit assignment method can be considered: after the completion of each iteration, the credit for the quantization error is assigned among the agents at the lowest level (i.e., agents that do not have any supervisors); those agents then distribute the credit received among their subordinates and themselves, and so on. Now, the agents can simultaneously adapt their number of Q-levels, as in the multiagent case.

In the original formulation of the resource allocation mechanism design problem [11], agents and supervisors repeatedly exchange messages until the *equilibrium* is reached. Then the outcome of the organization is a function of the equilibrium message. In this paper, we have considered a static message mechanism. It would be interesting to study the generalization of the proposed algorithm, or other methods, to the dynamic messaging model.

Finally, our algorithm is based on a number of heuristics. It would be nice if we could obtain some theoretical justification—through some convergence analysis—for those steps.

Acknowledgments

This work was done while P. Gupta was a graduate student at the Indian Institute of Science, Bangalore. The work of V. Borkar was supported by the Homi Bhabha Fellowship.

References

[1] S. C. Ahalt, A. K. Krishnamurthy, P. Chen, and D. Melton, "Competitive Learning Algorithms for Vector Quantization," *Neural Networks*, **3** (1990) 277–290.

[2] D. DeSieno, "Adding a Conscience to Competitive Learning," in *Institute of Electrical and Electronic Engineers International Conference on Neural Networks*, (1988) I117–I124.

[3] B. Fritzke, "Growing Cell Structures—A Self-organizing Network for Unsupervised and Supervised Learning," *International Computer Science Institute*, Technical Report TR-93-026, Berkeley, 1993.

[4] A. Gersho and R. M. Gray, *Vector Quantization and Signal Processing* (Kluwer, Boston, 1992).

[5] R. M. Gray, "Vector Quantization," *Institute of Electrical and Electronic Engineers Acoustics, Speech and Signal Processing Magazine*, **1**(2) (1984) 4–29.

[6] S. Grossberg, "Competitive Learning: From Interactive Activation to Adaptive Resonance," *Cognitive Science*, **11** (1987) 23–63.

[7] L. Hurwicz and T. Marschak, "Approximating a Function by choosing a Covering of its Domain and k points from its Range," *Journal of Complexity*, **4** (1988) 137–174.

[8] T. Kohonen, *Self-organization and Associative Memory* (second edition, Springer-Verlag, Berlin, 1988).

[9] Y. Linde, A. Buzo, and R. M. Gray, "An Algorithm for Vector Quantization Design," *Institute of Electrical and Electronic Engineers Transactions on Communications*, **28**(1) (1980) 84–95.

[10] S. P. Lloyd, "Least-square Quantization in PCM," *Institute of Electrical and Electronic Engineers Transactions on Information Theory*, **28**(2) (1982) 129–137.

[11] T. Marschak, and S. Reichelstein, "Communication Requirements for Individual Agents in Networks and Hierarchies," in *The Economics of Information Decentralization: Complexity, Efficiency and Stability*, edited by J. Ledyard (Kluwer, Boston, 1993).

[12] J. S. Rodrigues and L. B. Almeida, "Improving the Speed in Topological Maps of Patterns," *Proceedings of International Neural Networks Conference* (Paris, 1990).

[13] S. Sathiya Keerthi and B. Ravindran, "A Tutorial Survey of Reinforcement Learning," SĀDHANĀ: Indian Academy of Science Proceedings in Engineering Sciences, **19** (1994) 851–889.