STOCHASTIC VECTOR QUANTIZATION USING NEURAL NETWORKS

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF ELECTRICAL ENGINEERING
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Earl I. Levine

June 1996

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Robert M. Gray
(Principal Advisor)

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Bernard Widrow

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

David E. Rumelhart

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Martin Morf

Approved for the University Committee on Graduate Studies:

Abstract

In the real world we deal with information — still and moving images, speech, and sound, for example — in an analog form. In order to transmit this information over the Internet, or store it onto a hard disk, it must be digitized. Data compression is the process of representing the information digitally while striving to simultaneously minimize the number of bits needed to represent the data, maximize the quality of the reproduced data, and minimize the computational complexity required to do the encoding and decoding.

Nearest-neighbor vector quantization (VQ) is a simple but effective technique for performing data compression on a vector. A vector is encoded to the nearest neighbor in a finite codebook of possible reproduction vectors, that is, the reproduction with the least distortion from the original. Computation of the distortions during the search contributes most of the complexity. When the data to be quantized is a *sequence* of vectors, performance can be improved in theory by using state information kept coherently at the encoder and decoder to avoid encoding redundant information. Finite-state VQ (FSVQ) has a finite number of states, each with a codebook used for nearest-neighbor VQ; the next state in the sequence is a function of the current state and index. However, no techniques are known for optimal design of the next-state function, and allowing only finite states constrains performance.

A novel system which uses the regression properties of feedforward neural networks is presented. A neural network observing the original vector generates a probability distribution over the possible indices, from which one is chosen stochastically. Although this is suboptimal compared to nearest-neighbor VQ, there is no computation of distortions. Furthermore, unlike FSVQ, a state-feedback extension to this scheme with a continuous state vector and a second neural network (to generate the next state and reproduced vector) has an optimal training procedure. The main disadvantage of this scheme is the lengthy training time, typical of neural networks. Experimental results comparing this system to FSVQ, and to CELP (for speech coding) are presented.

Preface

This dissertation is dedicated to my wife Susan and to my parents, without whose love and patience I certainly never would have made it to this point. Susan has suggested that writing this preface should be easy since all I need to do is determine 200 ways to thank her. If only that were enough. How this incredible woman has managed to put up with my unusual working habits and the ensuing delinquency while simultaneously completing her own dissertation (and *hers* puts mine to shame) is beyond me. I suppose I *could* repay her by cancelling my plans to make this dissertation my retirement memoir and, after graduation, not depend solely on her superior bread-winning ability after all. (Hmm, I hope she doesn't read this.)

To my surprise but not theirs, the love for learning my parents have imbued has carried me through my many years at Stanford. Also, their emotional support has helped me deal with a lot. So I hope it does not come as a disappointment to them to hear that after 23 uninterrupted years as a student I will be giving the formal education system a break from me, at least for now. I also hope that this does not come as a disappointment to the formal education system.

I very much want to thank my dissertation reading committee who, because they have not read this preface before giving their approvals, are receiving my honest thanks and not simply a tribute designed to help along the approval process!

I suspect that by having been Professor Robert Gray's advisee for a mere four months or so preceding graduation, I have unintentionally set a worthless record; in retrospect I would much rather have been his student for my full time at Stanford. He has been a wonderful advisor during this short time, allowing me an undeservedly large amount of his precious time to give me advice on my work, and providing me

with enough financial support to graduate. In trying to determine why he has been so kind to me, the only conclusion I can draw is that he has a truly generous heart. I'm told if I ever make good in the "real world" I owe him an endowed chair — a more-than-fair deal for me!

It is quite fortunate for me that Professor Bernard Widrow is right here at Stanford and has been my secondary advisor. Since his innovative research throughout his career includes both neural networks and quantization, he is in the unique position of understanding well both sides of my work. I have really appreciated his wise advice, his caring attitude, and his sly sense of humor.

Like every other student who has known Professor David Rumelhart, I have been inspired by his example of open-mindedness and innovation to search out an untrodden research path in the spirit of his own work. I am truly lucky to have had the chance to get to know this brilliant man, to work in his PDP research group, and to engage him in such intellectual contests as ping-pong, volleyball, darts, ...

The conversations I have had with Professor Martin Morf about my dissertation work have been stimulating and very helpful. By asking questions from an unexpected perspective and introducing me to unanticipated ideas he has greatly enriched my understanding, for which I am grateful.

There are countless others who have advised me in one way or another during the course of my research. In particular I would like to thank Professor David Heeger, Dr. Ronjon Nag, Dr. Stephen Piche, members of the PDP research group past and present, and the members of Professor Gray's Signal Compression and Classification research group. I owe a special debt to Dr. Barry Andrews, without whom I surely would have taken another forty to fifty years to graduate. My failure to mention the many others who have helped me and inspired me along the way is, I'm afraid, a result of my poor memory and not any indication of my lack of gratitude!

I am especially thankful to Dr. Koichi Ejiri of the Ricoh Research and Development Center in Japan, who was most kind in going out of his way to provide an invaluable research and cultural experience during my term there. Also, thank you to Dr. Peter Hart and Dr. David Stork of the Ricoh California Research Center for financial support early in my time at Stanford.

Finally, I am eternally grateful to DNA, water, and the sun for being there when I needed them the most — without them this dissertation, and some other far less important things, would not have been possible.

Contents

List of Figures

Chapter 1

Introduction

Data compression is a fundamentally simple problem. One wishes to represent some data with a small number of bits, then reconstruct the data at high quality, and do all this with minimal computation. However the differing balance between these three factors allows many different techniques for data compression each to be "the best" for a different application. This openness to new methods makes the research field of data compression vibrant, diverse, and interesting.

This dissertation introduces a novel method for data compression which uses ideas from the field of neural networks. The technique is unlike any previous method for vector quantization, but traditional optimality principles are considered. This method is most useful when used with state-feedback and applied to data sources with memory, which describes nearly every data source of practical interest.

Chapter 2 introduces the general notion of regression, and the specific idea of the multi-layer feedforward neural network with its backpropagation training procedure. Chapter 3 introduces the vector quantization (VQ) approach to data compression and the concept of Lloyd-optimality, as well as relevant specific topics in VQ, including VQ with state feedback. In Chapter 4 the memoryless form of stochastic VQ using a feedforward neural network is presented. A training procedure is derived and optimality (or lack thereof) is discussed. Chapter 5 presents the state-feedback extension to the memoryless stochastic VQ of Chapter 4, along with a training procedure. This scheme has some immediately notable advantages over finite-state VQ (FSVQ, the classical extension of VQ to using state feedback), such as a continuous state vector and not calculating the distortion measure. Chapter 6 presents the results of two experiments with the system of Chapter 5: a comparison to FSVQ for benchmark data, and a comparison to CELP for low-rate speech coding. Finally, Chapter 7 summarizes the conclusions drawn from this work, and discusses future directions for research in this area.

Chapter 2

Regression and Neural Networks

2.1 Regression

A common problem in statistics and engineering is to estimate the value of a variable (or set of variables) from the value of another variable (or set of variables), when there is a relationship between these variables. An example would be to estimate the height and age of people from their weight and shoe size. Such problems are known as "data fitting" or "data modeling" problems, and there are other names for this type of problem in different fields of study.

Implicit in data fitting problems is a performance measure appropriate for the particular data domain. This measure is often a function measuring the error between the estimated values and the "actual" values. A common example of such an error function is mean squared error (MSE), which is the squared difference between estimated and actual values.

If the problem domain is one for which there is a well-developed theory about the relationship underlying the variables, then the best way to fit the data is often the direct application of the theory. On the other hand, if little is known in theory about the problem, but a lot of sample data is available, then regression techniques can be very useful.

Regression is the use of a parametric function to do the estimation. The values of the variables to be estimated (outputs) are taken to be a function of the observable variables (inputs) and of a set of adjustable parameters. The success of a regression model depends on the type of function, the number of parameters, and the values of the parameters.

Let all the input variables be collected into vector $\mathbf{x}$ and all the outputs of the regression function be vector $\hat{\mathbf{y}}$, an estimate of the true value of $\mathbf{y}$. The simplest case of regression "function" is a constant estimate,

$$\hat{\mathbf{y}} = \mathbf{y}_{\text{constant}}, \tag{2.1}$$

in which case the only parameters of the function are $\mathbf{y}_{\text{constant}}$. A commonly used regression function is a linear fitting,

$$\hat{\mathbf{y}} = \mathbf{W}\mathbf{x}, \tag{2.2}$$

where the parameters are in matrix $\mathbf{W}$. Affine fitting is linear fitting with a constant offset, that is,

$$\hat{\mathbf{y}} = \mathbf{W}\mathbf{x} + \mathbf{b}, \tag{2.3}$$

where now the parameters are in matrix $\mathbf{W}$ and vector $\mathbf{b}$. A quadratic fit consists of a weighted sum of all polynomial terms of order 2 or lower. If the vector $(\mathbf{x}, 1)$ is the vector $\mathbf{x}$ concatenated with a 1 as the final component, then the quadratic fit can be expressed as

$$\hat{\mathbf{y}} = \begin{pmatrix} (\mathbf{x},1)^T \mathbf{W}_1 (\mathbf{x},1) \\ (\mathbf{x},1)^T \mathbf{W}_2 (\mathbf{x},1) \\ \vdots \\ (\mathbf{x},1)^T \mathbf{W}_K (\mathbf{x},1) \end{pmatrix}, \tag{2.4}$$

where $\mathbf{x}^T$ here denotes the transpose of $\mathbf{x}$, and where $\mathbf{W}_k$ is the kth (square) matrix of parameters for this second-order polynomial, and there are K components of $\hat{\mathbf{y}}$. For $\hat{\mathbf{y}}$ of dimension K and $\mathbf{x}$ of dimension L, a polynomial fit of general order M may be expressed as

$$\hat{y}_k = \sum w_{k,m_1,m_2,\ldots,m_L} x_1^{m_1} x_2^{m_2} \cdots x_L^{m_L}, \qquad \forall k \tag{2.5}$$

where the sum is over all combinations of nonnegative integers m_l such that

$$\sum_{l=1}^{L} m_l \leq M, \tag{2.6}$$

and all the $w_{k,m_1,m_2,\ldots,m_L}$ values are parameters of the fit. The number of possible regression functions is unlimited.

For a given choice of regression function, the parameter values must be chosen to optimize performance. A common technique is to gather a number of co-occurrence values of $\mathbf{x}$ and $\mathbf{y}$, then from that data set randomly select a subset to be used, along with a training procedure, to train the parameter values. This data set is known as the "training set." The training procedure, if one exists, depends on the type of regression function. Another set of data, randomly chosen from the set of all data exclusive of training set data, is used to evaluate the performance after or during the training procedure. This data set is known as the "testing set."

A regression function which, after training, models the training set very well but models the testing set poorly is said to "overfit" the training data. Figures 2.1 and 2.2 show an example of an overfitting situation where x and y are scalars. In Figure 2.1, points representing the testing set (generated for this example by a 5-degree polynomial function plus Gaussian noise) were used to find the optimal parameters for a 15-degree polynomial fit, shown as the light gray line. Although this fit appears to be very good in Figure 2.1, a comparison in Figure 2.2 of this fit with some more data points not in the training set — a testing set — shows that the fit is not so suitable for novel data. Often overfitting is a symptom of a regression function with too many parameters, also known as "too many degrees of freedom." Using a "smaller" function, that is fewer parameters, may mitigate the overfitting, as may choosing a different type of regression function. The possibility of overfitting is one reason that the testing set must be used to evaluate the performance of a regression function, rather than the training set.

The opposite problem, "underfitting," is also possible. Underfitting is the condition of poorly modeling both the training set and the testing set. Again, Figures 2.1 and 2.2 show an example of an underfitting situation. The medium-gray line shows

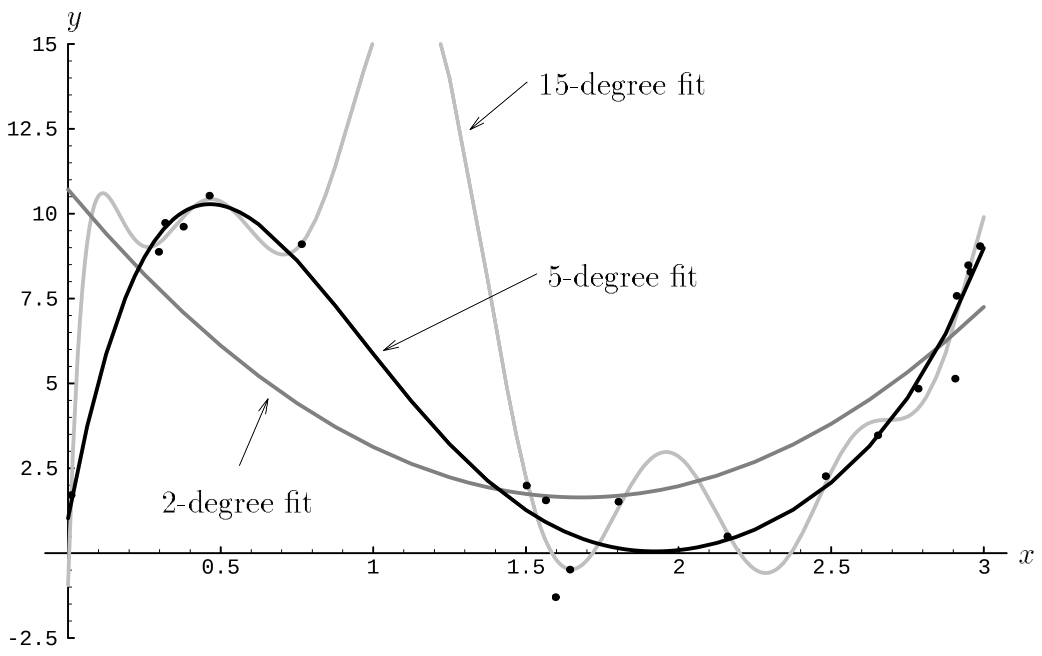

Figure 2.1: Three Polynomial Fits for a Training Data Set

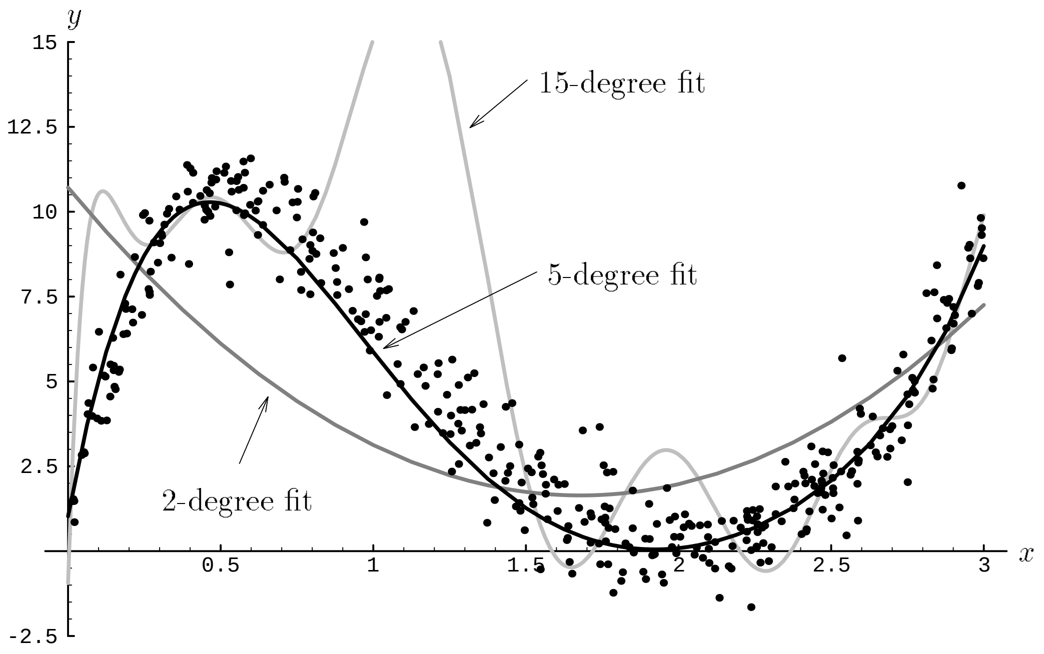

Figure 2.2: Three Polynomial Fits Compared for a Testing Data Set

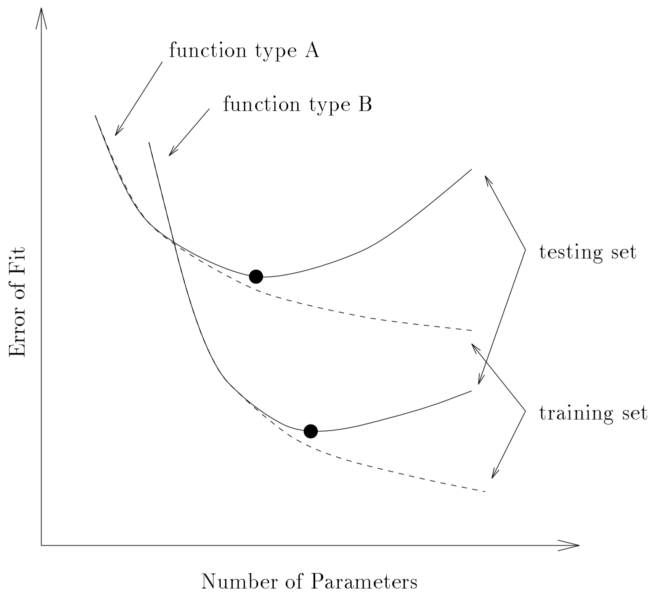

Figure 2.3: Comparison of Function Types and Number of Parameters for Regression

a 2-degree polynomial fit to the training data, which is in fact an equally poor fit for both training set and testing set. Often underfitting is a symptom of a regression function with too few parameters, also known as "too few degrees of freedom."

The best number of parameters to use for a polynomial fit of the data in the example of Figures 2.1 and 2.2 happens to be 5, as shown by the black line in both figures. While the 5-degree polynomial does not fit the training data as well as the 15-degree polynomial, the fit of the testing data — which is the essential criterion — is much better than the 15-degree fit and the 2-degree fit.

Figure 2.3 illustrates the choice of type of function and number of parameters. For a given type of function, the optimal number of parameters is the one for which testing set performance is best, shown as the black dots. This often occurs at nearly the number of parameters for which overfitting begins to be a problem. Then, to decide between multiple types of regression function, the type is chosen which has the best performance at its optimal number of parameters. In the example shown, function type B should be used with the number of parameters for the best testing set fit (the black dot). Although in practice it is usually not possible to try out all numbers of parameters for a given function type, and it is certainly not possible to try out all function types, this gives a general criterion for choosing one combination of function type and number of parameters over another.

2.2 Linear Regression

One common type of regression problem requires minimizing the mean squared error of a linear or affine regression function. This is commonly known as the *linear least squares* fitting problem.[30] Note in the following discussion that affine regression may be made to appear like linear regression by allowing $\mathbf{x}$ an extra component which is permanently fixed at 1; therefore these "linear" methods may also be used for affine regression. Two useful methods can be employed to optimize the parameter values for this situation.

2.2.1 Wiener-Hopf

The first is the "Wiener-Hopf" method, which finds the optimal matrix of parameters

$$\begin{aligned}\mathbf{W}^* &= \underset{\mathbf{W}}{\operatorname{argmin}}\, \mathrm{E}[\|\mathbf{y} - \hat{\mathbf{y}}\|^2] && (2.7)\\ &= \underset{\mathbf{W}}{\operatorname{argmin}}\, \mathrm{E}[\|\mathbf{y} - \mathbf{W}\mathbf{x}\|^2], && (2.8)\end{aligned}$$

where $\|\mathbf{z}\|^2$ denotes the sum of squared components of a vector $\mathbf{z}$. The error is a quadratic function of $\mathbf{W}$, and is minimized by

$$\mathbf{W}^* = \mathbf{P}\mathbf{R}^{-1}, \tag{2.9}$$

where $\mathbf{R}$ is the autocorrelation of $\mathbf{x}$,

$$\mathbf{R} = \mathrm{E}[\mathbf{x}\mathbf{x}^T], \tag{2.10}$$

and where $\mathbf{P}$ is the cross-correlation of $\mathbf{x}$ and $\mathbf{y}$,

$$\mathbf{P} = \mathrm{E}[\mathbf{y}\mathbf{x}^T]. \tag{2.11}$$

Wiener-Hopf has the advantage of being a one-step method (as opposed to an iterative method) . Wiener-Hopf can fail if the matrix $\mathbf{R}$ is singular and therefore cannot be inverted. Similarly, if $\mathbf{R}$ is nearly singular, the inverse can be "poorly conditioned," meaning that there is a very high sensitivity to numerical precision. This is the multidimensional analog to the scalar case of division by a number which is nearly zero. Thus techniques such as ridge regression must be used to modify the parameters.[23]

2.2.2 Least Mean Squares (LMS)

The second method often used is the "least mean squares" (LMS) method, a stochastic gradient descent method also known as the Widrow-Hoff method.[34]

In general, gradient descent [22] is an optimization method for finding the value of parameters $\mathbf{W}$ which minimizes an objective function $\varepsilon(\mathbf{W})$, a continuous and differentiable function of $\mathbf{W}$. An initial value for $\mathbf{W}$ is selected (for example, randomly) and then $\mathbf{W}$ is iteratively adjusted along the gradient of the objective ε by

$$\mathbf{W}_{\text{next}} = \mathbf{W}_{\text{current}} - \mu \frac{\partial \varepsilon}{\partial \mathbf{W}_{\text{current}}}, \tag{2.12}$$

where μ is a positive constant known as the "learning rate." In a small enough neighborhood of $\mathbf{W}_{\text{current}}$, $\varepsilon(\mathbf{W})$ appears as a hyperplane which slopes down fastest in the direction opposite the gradient $\frac{\partial \varepsilon}{\partial \mathbf{W}_{\text{current}}}$. Therefore a small enough value of μ will cause each iteration to be a step downward in the direction of steepest descent of ε. After a certain number of iterations, gradient descent will eventually reach a fixed point at a local minimum.

Gradient descent is well suited to the linear least squares problem, since the error

$$\varepsilon(\mathbf{W}) = \mathrm{E}[\|\mathbf{y} - \hat{\mathbf{y}}\|^2] = \mathrm{E}[\|\mathbf{y} - \mathbf{W}\mathbf{x}\|^2] \tag{2.13}$$

is a quadratic function of $\mathbf{W}$. This implies that the error surface is a smooth quadratic "bowl" with a single minimum, therefore gradient descent can find the global minimum. In practice, the choice of value of μ resembles the overdamping / underdamping / critical damping scenario commonly seen in engineering problems.

Rather than compute the exact gradient of the error as in Equation 2.13, LMS is stochastic in that a single novel training pair sample $\{\mathbf{x}_k, \mathbf{y}_k\}$ is used to determine an estimate of the gradient for each iteration k. That is, since the true error is an expectation over all training pairs $\{\mathbf{x}_k, \mathbf{y}_k\}$,

$$\begin{aligned} \varepsilon &= \mathrm{E}_k[\|\mathbf{y}_k - \mathbf{W}\mathbf{x}_k\|^2] &(2.14)\\ &= \mathrm{E}_k[\varepsilon_k], &(2.15)\end{aligned}$$

then ε can be estimated stochastically by ε_k. Similarly, $\frac{\partial\varepsilon}{\partial\mathbf{W}}$ can be estimated stochastically by

$$\begin{aligned} \frac{\partial\varepsilon_k}{\partial\mathbf{W}} &= \frac{\partial[(\mathbf{y}_k - \mathbf{W}\mathbf{x}_k)^T(\mathbf{y}_k - \mathbf{W}\mathbf{x}_k)]}{\partial\mathbf{W}} &(2.16)\\ &= -2(\mathbf{y}_k - \mathbf{W}\mathbf{x}_k)\mathbf{x}_k^T. &(2.17)\end{aligned}$$

The iterative learning rule then becomes

$$\mathbf{W}_{\text{next}} = \mathbf{W}_{\text{current}} + 2\mu(\mathbf{y}_k - \mathbf{W}\mathbf{x}_k)\mathbf{x}_k^T. \tag{2.18}$$

One of the most common applications of the LMS algorithm is for adaptive linear filters, in which case the training values $\{\mathbf{x}_k\}$ are successive states of a tapped delay line, and $\mathbf{W}$ represents the filter weights.

LMS has some advantages over Wiener-Hopf. First, there is no matrix inversion involved. Therefore there are no matrix singularity or near-singularity problems. Also, depending on the number of iterations needed to converge to near the optimum, and the number of components of the input and output vectors, LMS may have less complexity than Wiener-Hopf. Second, while Wiener-Hopf is a one-time solution for a given training set, LMS can continuously adapt the parameters if the optimal values are changing over time, due to a change in the process being modeled. For example, LMS may be used as an adaptive equalizer to compensate for the filtering effected by a radio channel link between a mobile telephone and a base station receiver. As the position of the mobile telephone changes during the connection, a change in the channel requires continuous adaptation of the equalizer.

2.3 Nonlinear Regression

It is possible to build on the linear/affine regression function by applying some nonlinear function to the vector result of the linear or affine operation. This nonlinear function could be considered fixed, or could depend on parameters which are included in the set of adjustable parameters of the entire regression function. If the class of "nonlinear" functions allowed includes linear multiplication by the identity matrix (that is, just passing the inputs to the outputs) then clearly the linear/affine function is a subset of all such allowable regression functions. Therefore performance can only be improved or kept the same by considering the addition of a nonlinear function.

2.3.1 Perceptrons

One type of problem which suggests a particular choice of nonlinear function is the modeling of a function with binary outputs. That is, the correct outputs of the function can only be ones or zeros. Some such functions (for example, binary logic functions like AND, OR and exclusive-or XOR) have binary inputs as well.

For binary output problems, one type of regression function popularized in the 1960's by Rosenblatt [28] was a "perceptron" model. This is the function

$$\hat{\mathbf{y}} = \mathbf{f}_{\text{perceptron}}(\mathbf{x}) = \frac{1}{2}\big(1 + \mathbf{sgn}(\mathbf{W}\mathbf{x} + \mathbf{b})\big), \tag{2.19}$$

where the **sgn** function is the component-wise application of the signum threshold function:

$$\operatorname{sgn}(z) = \begin{cases} -1 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0. \end{cases} \tag{2.20}$$

This threshold function forces the output to be binary.[1] For each output component, the perceptron defines a hyperplane in the space of the input vector (whose placement is determined by the parameter values) which is the boundary between the input values which will produce a zero-output, and those which will produce a one-output.

[1] If output values of -1 and 1 are desired (rather than 0 and 1), the signum function can be used directly without scaling, as is done for the ADALINE model.[32]

The perceptron is trained in the following way: the affine function $\mathbf{W}\mathbf{x} + \mathbf{b}$ is trained to minimize the MSE between its own output and the desired binary function (scaled so that zeroes become -1),

$$\|(2\mathbf{y} - 1) - \mathbf{W}\mathbf{x} + \mathbf{b}\|^2 \tag{2.21}$$

using LMS or possibly Wiener-Hopf; then the trained parameters from the affine model are used in the perceptron function, Equation 2.19. Note that the training is performed on a different function than will actually be used, therefore this training procedure does not find the optimal set of parameters. Despite suboptimal training, the perceptron is capable of modeling some problems well.

2.3.2 Multi-Layer Perceptrons and Backpropagation

However, there is an entire class of problems for which it can be shown that the perceptron is simply incapable of a correct model, or sometimes, even an acceptable model. Such problems are so-called "non-linearly-separable" (NLS) problems. One of the simplest examples of an NLS problem is the 2-input XOR function. A single linear boundary (the input has 2 components; a hyperplane in 2 dimensions is a line) is incapable of dividing the input space into inputs which should produce a zero-output, and those which should produce a one-output. Since perceptrons are only capable of a single hyperplane boundary, this problem cannot be modeled correctly by a perceptron.

In order to solve NLS binary problems, it is necessary to consider the "feedforward[2] multi-layer perceptron" (MLP) which is described by the function:

$$\hat{\mathbf{y}} = \mathbf{f}_{\text{perceptron},L}(\mathbf{f}_{\text{perceptron},L-1}(\cdots \mathbf{f}_{\text{perceptron},2}(\mathbf{f}_{\text{perceptron},1}(\mathbf{x}))\cdots)), \tag{2.22}$$

where L is known as the number of layers.[3] However, a training procedure for this type of function is unknown. One idea that was considered for training the MLP was

[2]The name "feedforward" distinguishes these networks (in which the output of a layer may only feed forward into the next layer) from networks in which the output of one layer may feed back into the inputs of any arbitrary layer. Such networks are often more complex dynamic systems, and are not considered in this dissertation.

[3]For historical reasons, the input is sometimes also counted as a layer, in which case the number of layers is considered to be $L + 1$.

to train using the function

$$\hat{\mathbf{y}} = \frac{1}{2}\big(1 + \mathbf{sgn}(\mathbf{b}_L + \mathbf{W}_L(\mathbf{b}_{L-1} + \mathbf{W}_{L-1}(\cdots \mathbf{b}_2 + \mathbf{W}_2(\mathbf{b}_1 + \mathbf{W}_1(\mathbf{x}))\cdots))))\big) \quad (2.23)$$

which only has one layer of thresholding. However, this function simply collapses to the usual type of perceptron

$$\hat{\mathbf{y}} = \frac{1}{2}\big(1 + \mathbf{sgn}(\mathbf{b}_{\text{all}} + \mathbf{W}_{\text{all}}(\mathbf{x}))\big) \quad (2.24)$$

and is therefore again unable to solve NLS problems. This inability to solve NLS problems was claimed by Minsky and Papert [24] in 1969 to be a fatal flaw justifying the abandonment of further research in perceptrons. It took almost 20 years before Rumelhart et al. [29] popularized a way to overcome this problem and make useful general regression functions from MLPs.[4] [5]

The key to overcoming the trainability hurdle for MLPs is the use of a continuous thresholding function in place of the hard-threshold function in Equation 2.22, for example, the sigmoid function

$$\text{sgm}(z) = \frac{1}{1 + e^{-z}}. \quad (2.25)$$

Then, assuming that the error is a continuous and differentiable function of the regression function's output, [6] the gradient of the error with respect to all the parameters can be calculated for given parameter values. This allows the use of a gradient descent minimization method to train the parameters.

When $L = 2$, the top layer is often called the *output layer* and the lower layer is called the *hidden layer*. As noted above this may also be referred to as a three-layer network (if one likes to count the input as a layer). It is a surprising but powerful result (shown by Cybenko [5] and others) that these MLPs in fact need only $L = 2$ in order to fit any function with desired accuracy, given enough components in the hidden layer. However it is not known in general how much is "enough," therefore

[4]Although Rumelhart et al. are known for independently discovering and popularizing the ideas in the late 1980's, credit for first publication of these concepts in 1974 goes to Werbos[31].

[5]For an discussion of the interesting history of neural networks research see [14] or [33].

[6]This is true for many commonly used error functions, for example MSE.

often only experimentation can determine the correct size. A general expression for an MLP with $L = 2$ is

$$\hat{\mathbf{y}} = \mathbf{f}_\mathrm{u}\big(\mathbf{b}_\mathrm{u} + \mathbf{W}_\mathrm{u}\mathbf{f}_\mathrm{l}(\mathbf{b}_\mathrm{l} + \mathbf{W}_\mathrm{l}\mathbf{x})\big) \tag{2.26}$$

where $\mathbf{f}_\mathrm{u}$ and $\mathbf{f}_\mathrm{l}$ are the nonlinear vector functions of the upper and lower layers, respectively (such as component-wise application of the sigmoid or hyperbolic tangent functions), and $\mathbf{W}_\mathrm{u}$, $\mathbf{W}_\mathrm{l}$, $\mathbf{b}_\mathrm{u}$, and $\mathbf{b}_\mathrm{l}$ are the parameters which will be trained.

Returning to a training procedure, in order to use gradient descent, a method for computing the gradient must be found. The key to backpropagation is the use of the chain rule of calculus to determine the error gradient at the lower layer in the network. For a single training pattern $\{\mathbf{x}, \mathbf{y}\}$, the gradient at the network's output is computed from the error function

$$\frac{\partial \varepsilon}{\partial \hat{\mathbf{y}}} = \frac{\partial \varepsilon(\mathbf{y}, \hat{\mathbf{y}})}{\partial \hat{\mathbf{y}}}. \tag{2.27}$$

Then the gradient at the input of the upper nonlinear function, which is also the gradient at the vector $\mathbf{b}_\mathrm{u}$, is found as

$$\frac{\partial \varepsilon}{\partial \mathbf{b}_\mathrm{u}} = \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{b}_\mathrm{u}} \frac{\partial \varepsilon}{\partial \hat{\mathbf{y}}} = \frac{\partial \mathbf{f}_\mathrm{u}}{\partial \mathbf{b}_\mathrm{u}} \frac{\partial \varepsilon}{\partial \hat{\mathbf{y}}}, \tag{2.28}$$

where $\frac{\partial \mathbf{f}_\mathrm{u}}{\partial \mathbf{b}_\mathrm{u}}$ comes from differentiation of that function. For the sigmoid, a shortcut for computing the derivative of this function is

$$\frac{\partial \mathrm{sgm}(z)}{\partial z} = \frac{e^{-z}}{(1 + e^{-z})^2} = \mathrm{sgm}(z)(1 - \mathrm{sgm}(z)), \tag{2.29}$$

which is useful because $\mathrm{sgm}(z)$ has already been computed. (Similar shortcuts exist for hyperbolic tangent and other nonlinear functions commonly used in neural networks.) The continued application of the chain rule yields

$$
\begin{aligned}
\frac{\partial \varepsilon}{\partial \mathbf{W}_\mathrm{u}} &= (\mathbf{f}_\mathrm{l}(\mathbf{b}_\mathrm{l} + \mathbf{W}_\mathrm{l}\mathbf{x})) \left(\frac{\partial \varepsilon}{\partial \mathbf{b}_\mathrm{u}}\right)^T, && (2.30)\\
\frac{\partial \varepsilon}{\partial \mathbf{f}_\mathrm{l}()} &= \mathbf{W}_\mathrm{u}^T \frac{\partial \varepsilon}{\partial \mathbf{b}_\mathrm{u}}, && (2.31)\\
\frac{\partial \varepsilon}{\partial \mathbf{b}_\mathrm{l}} &= \frac{\partial \mathbf{f}_\mathrm{l}()}{\partial \mathbf{b}_\mathrm{l}} \frac{\partial \varepsilon}{\partial \mathbf{f}_\mathrm{l}()}, && (2.32)\\
\frac{\partial \varepsilon}{\partial \mathbf{W}_\mathrm{l}} &= \mathbf{x} \left(\frac{\partial \varepsilon}{\partial \mathbf{b}_\mathrm{l}}\right)^T, \text{ and} && (2.33)\\
\frac{\partial \varepsilon}{\partial \mathbf{x}} &= \mathbf{W}_\mathrm{l}^T \frac{\partial \varepsilon}{\partial \mathbf{b}_\mathrm{l}}, && (2.34)
\end{aligned}
$$

which gives the gradient over all parameters as well as the gradient at the input. Since there are usually multiple vectors $\mathbf{x}$ in the training set, the expected value of the gradient over the entire set is used for gradient descent. This is simply found as an average (weighted by any prior probabilities over the set) of the gradients computed for each individual $\{\mathbf{x}, \mathbf{y}\}$. The use of the chain rule to propagate the error derivative "back" (toward the inputs) through the function is known as "backpropagation", a name given by Rumelhart et al.

Backpropagation, like LMS, is sensitive to a good choice of learning rate. As mentioned previously, gradient descent can eventually find a local minimum of error. However a *global* minimum[7] is most desirable. As the number of parameters increases the likelihood of settling in a non-global minimum becomes increasingly smaller; non-global minima are usually only a concern for very small numbers of parameters. Of course this is all dependent on the specific error function. One technique that can improve the chances of avoiding non-global minima is adding a controlled amount of noise to the gradient during training.[15]

Perceptrons and multilayer perceptrons are members of a greater class of systems historically known as "neural networks" because they were originally inspired by biophysical theories about how networks of neurons interact in the brain. This thesis is not concerned with the biological implications of such systems, only their use as

[7]Multiple global minima are possible, therefore there might not be such a thing as "the" global minimum.

regression functions.

2.3.3 Probabilistic Interpretation

Consider again problems for which one wishes to model a binary function using a MLP with the usual sigmoid output function. There seems to be an inconsistency between the continuous output required for backpropagation training and the discrete output required by the desired function. One common method is to first train the MLP to minimize the squared error between its output and the desired binary output, then once trained, replace the continuous sigmoid output function with a hard threshold function $\frac{1}{2}(1 + \mathrm{sgn}())$. Just as in the usual single-layer perceptron case, however, this is suboptimal training since the training is done for a different function than will actually be used.

A probabilistic interpretation suggested by Rumelhart[3] is able to resolve the inconsistency between continuous MLP output and binary desired output. The sigmoid output $\hat{y}$ is taken to mean the probability that a corresponding binary output b is one-valued (and not zero-valued). The use of a continuous sigmoid output function is therefore consistent with this probability, which may be any value in the range 0 to 1. The error to be minimized may be derived by considering the likelihood P that the output b (whose probability of being one-valued is $\hat{y}$, the MLP output) is correct, given that the desired value of b is y,

$$P = \begin{cases} \hat{y} & \text{if } y = 1, \\ 1 - \hat{y} & \text{if } y = 0. \end{cases} \tag{2.35}$$

This likelihood should be maximized. Because of the restrictions on the values of $\hat{y}$ and y, one possible re-expression of P is

$$P = 1 - |y - \hat{y}|. \tag{2.36}$$

Since $P < 1$, the quantity

$$\tilde{P} = (1 - P)^2 \tag{2.37}$$

$$= (y - \hat{y})^2 \tag{2.38}$$

is a monotonically decreasing function of P, therefore minimizing $\tilde{P}$ is exactly equivalent to maximizing P. So it can be seen that minimizing the usual squared error measure also maximizes the likelihood that the probabilistically interpreted output is correct.

In general one might wish to model a function which can only have one of N possible output values. The above binary case is the specific case of such a problem for $N = 2$. For larger values of N, one can devise an MLP with a set of N outputs $\hat{y}_i$, each corresponding to the probability that each discrete output value i should be the one chosen. This requires the constraints

$$0 \leq \hat{y}_i \leq 1 \qquad \forall i, \text{ and} \tag{2.39}$$

$$\sum_{i=1}^{N} \hat{y}_i = 1. \tag{2.40}$$

To accomplish this, Rumelhart suggests the use of the "normalized exponential" nonlinear function[8] for the output layer,

$$\hat{\mathbf{y}} = \mathbf{f}_{\text{normexp}}(\mathbf{z}) \tag{2.41}$$

$$= \frac{1}{\sum_{i=1}^{N} \exp(z_i)} \begin{pmatrix} \exp(z_1) \\ \exp(z_2) \\ \vdots \\ \exp(z_N) \end{pmatrix}. \tag{2.42}$$

As before, the error to be minimized may be derived by considering the likelihood P that the chosen output i (whose probability of being chosen is $\hat{y}_i$, the corresponding MLP output) is correct, given that the correct output value choice is j,

$$P = \hat{y}_j. \tag{2.43}$$

This likelihood should be maximized. One possible way to re-express P is

$$P = \prod_{i=1}^{N} \hat{y}_i^{y_i}, \text{ where } y_i = \begin{cases} 1 & \text{if } i = j, \\ 0 & \text{if } i \neq j. \end{cases} \tag{2.44}$$

[8]This is reminiscent of the "tilted distribution" seen in source coding theory. [12]

Note that for a given correct choice j, the term

$$\sum_{i=1}^{N} y_i \log(y_i)$$

is constant. Then, since

$$\begin{aligned}\tilde{P} &= -\log(P) + \sum_{i=1}^{N} y_i \log(y_i) & (2.45)\\ &= \sum_{i=1}^{N} y_i \log(\frac{y_i}{\hat{y}_i}) & (2.46)\end{aligned}$$

is a monotonically decreasing function of P, minimizing $\tilde{P}$ is exactly equivalent to maximizing the likelihood P. This form of $\tilde{P}$ is known in information theory[4] as the Kullback-Leibler distance, the relative entropy, and the cross-entropy.

2.4 Regression with State Memory

Sometimes the inputs and desired outputs of a regression function are actually drawn from sequences $\mathbf{x}_t$ and $\mathbf{y}_t$. Of course the sequential nature of the data could be ignored and each $\mathbf{y}_t$ could be separately fit. However, if there is some correlation between successive values in the sequence, then in theory performance can be improved by keeping information about the past values of the sequence in some internal state memory.

One way to do this is to use two regression functions

$$\begin{aligned}\hat{\mathbf{y}}_t &= \mathbf{f}_y(\mathbf{x}_t, \mathbf{s}_t), \text{ and} & (2.47)\\ \mathbf{s}_{t+1} &= \mathbf{f}_s(\mathbf{x}_t, \mathbf{s}_t), & (2.48)\end{aligned}$$

where $\mathbf{s}_t$ is the internal state memory. Ideally both regression functions can be jointly optimized for best performance in estimating the $\hat{\mathbf{y}}_t$ sequence. As long as the $\mathbf{f}_y()$ could be trained to "ignore" the state (for example, setting to 0 a matrix parameter which multiplies the state vector), then allowing the use of state memory can never force the regression performance to worsen.

For at least one point in the sequence the state must be initialized to a "reset" value $\mathbf{s}_{\mathrm{init}}$, which could be chosen arbitrarily, or could be included in the list of parameters to train. The state $\mathbf{s}_t$ might be a vector, scalar, or even a discrete quantity taking on only one of N possible values. Is is important to note, however, that this state is always an internal "hidden" quantity and therefore it is not necessary to impose an interpretation on it; rather it can be thought of as an opportunity for the regression system to use memory however it wishes in order to optimize performance.

The functions $\mathbf{f}_y()$ and $\mathbf{f}_s()$ could be chosen to be MLPs, in which case the state is usually a continuous vector. In practice, the two functions may share a common hidden layer term. The weights of $\mathbf{f}_y()$ and $\mathbf{f}_s()$, as well as the value of $\mathbf{s}_{\mathrm{init}}$ (if desired) can be jointly trained using gradient descent. This involves starting with the error gradient at the $\mathbf{f}_y()$ outputs and propagating the computation of error gradient recursively back in t toward all the inputs. This method is known in the neural network field as backpropagation-through-time (BPTT)[3, 15].

Chapter 3

Data Compression and Vector Quantization

Every day we absorb information with our senses: still and moving images, speech, and sound. This information is inherently analog. Yet our modern media for communications and storage of information — for example, the Internet or computer hard disks — are digital. Therefore, in order to store or send such information using these media, the information must be converted into a digital representation (encoded) and converted back to a reconstruction which resembles the original information (decoded).

Data compression is the process of doing this in the most efficient way possible. Specifically, there are three simultaneous goals for data compression:

- Minimize the number of bits (or symbols) needed for the digital representation.
- Maximize the quality of the decoder's reconstruction of the data; or, stated differently, minimize the distortion incurred by the encoding and decoding process.
- Minimize the computational complexity required to perform the encoding and decoding.

The exact application of the data compression will determine the relative importance of these three factors.

The number of symbols, or bits (if the number of symbols is a power of 2), is generally known as the *rate*. For example, a system to encode sound might be said to have a rate of 8000 bits per second, or a system to encode a still image might have a rate of 0.5 bits per pixel (picture element). When storing data on a limited-size medium such as a hard disk, a lower rate allows more information to be kept on the disk. When transmitting the information, digital links of finite bit rate such as modems take less time to transmit information which has been compressed to a lower rate. If digital transmission over a radio channel is desired, then compressing the information to a lower rate allows less bandwidth of the radio spectrum (which is a finite resource) to be used, or less complex modulation and demodulation schemes to be used. For digital transmission channels of a finite bit rate, telephony applications (real-time communication of speech or video) might require that the data be compressed to a rate below a certain minimum threshold.

A data compression system must also provide a faithful reproduction of the original data from the decoder. Maximizing the quality is equivalent to minimizing a *distortion measure* between the original data and its reproduction. The correct type of distortion measure is dependent on the type of data and how the decoded data is to be used. This is discussed in more detail in Section 3.4.

In fact there is a direct relationship between rate and distortion. In information theory, Shannon's theory of rate-distortion bounds shows that the limit of attainable average distortion is a decreasing function of rate.[4, 9] In other words, even the best possible coding system for a given information source can only decrease distortion by increasing rate, or conversely, decrease rate by increasing distortion. This fundamental result suggests that even practical data compression must trade off one for the other.

In a practical system, computational complexity of the encoding and decoding algorithms is a concern. Complexity is hard to measure exactly because it depends so much on the implementation of the algorithm in hardware or software, but gross differences in complexity between two algorithms are convincing regardless of implementation. In general computation is a tradeoff between the following factors:

Time A given processor can only execute an algorithm in a given amount of time. It may be possible to spend more money to buy or build a faster processor, or more electrical power may be consumed in order to run the processor at a faster clock speed (to a limit), thereby speeding the algorithm's run time. Real-time applications require that the algorithm be run within a certain time constraint and are inflexible in this respect.

Money As mentioned above, money can buy a faster processor. Sometimes buying more memory allows an algorithm to be run faster. To increase the electrical power, higher-capacity batteries or more batteries (for a portable device) or cooling systems to remove waste heat and allow faster clock speeds could be purchased.

Power Electrical power is especially a concern for portable battery-operated devices which have limited energy on which to operate. A processor which uses more power may require a cooling system to allow proper error-free operation as well as to avoid heat-damaging the system.

Therefore reducing the algorithmic complexity allows some combination of these factors to be improved.

The tradeoff between the three major factors — rate, distortion, and complexity — and their relative importance, so different for different applications, is what allows so many data compression methods to be useful. One method would be considered superior to another only if it were better in some of the three factors and not significantly worse on the others. Therefore the application should determine the choice of method by weighing these factors.

A fourth factor which is sometimes important for data compression is the latency or delay. For example, if a speech-coding system must acquire an entire segment of one second length before any encoding can be performed on that segment, then there is at least a one second delay (plus the computation time and transmission time) from

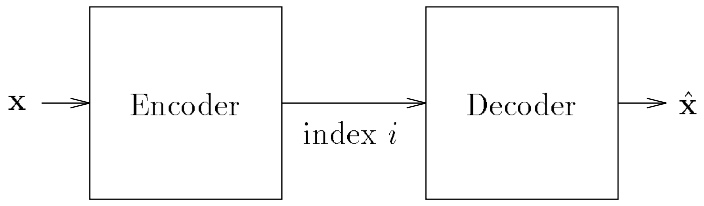

Figure 3.1: Vector Quantization

the acquisition of the speech by the encoder until the decoder can begin to decode that segment. In a real-time application such as a telephone call using this speech coding, long delay times may be annoying to people attempting to converse, and can also create echo problems which must be corrected by additional equipment.[11]

3.1 Lloyd-Optimal Vector Quantization

Suppose that the data to be compressed is represented by a vector $\mathbf{x}$. The encoder will produce an index i which is an integer from 1 to N, where N is the number of symbols; put another way, $\log_2 N$ is the rate in bits per vector. Let $\hat{\mathbf{x}}$ be the reconstruction of $\mathbf{x}$ which the decoder produces from the index i. This general system, depicted in Figure 3.1, is known as *vector quantization* or VQ. Implicit in the system is the measure of distortion (or error) between $\mathbf{x}$ and $\hat{\mathbf{x}}$ which is calculated using a given function $d(\mathbf{x}, \hat{\mathbf{x}})$.

Classical memoryless VQ, also referred to as "plain VQ" or "vanilla VQ," is based on the use of a *codebook*. This codebook is simply a table of N possible values for $\hat{\mathbf{x}}$. For each value of i, the decoder operates by looking up the corresponding value in the codebook $\hat{\mathbf{x}}_i$, and presenting that value as the output $\hat{\mathbf{x}}$.

The two Lloyd optimality conditions suggest how to use the codebook to encode, and also suggest one way to train the codebook. As pointed out by Lloyd,[21] these two conditions are necessary for an optimal quantizer:

- The encoder is optimal, given the decoder.

- The decoder is optimal, given the encoder.

When the decoder is a codebook, these simple conditions manifest themselves as, respectively, the *nearest neighbor condition* and the *centroid condition*.

For a given decoder, the encoder is optimal if the index i chosen will be the one that minimizes the distortion $d(\mathbf{x}, \hat{\mathbf{x}})$ between the original vector $\mathbf{x}$ and the decoder's reconstruction $\hat{\mathbf{x}}_i$. That is, each time a vector $\mathbf{x}$ is to be encoded, for each possible index i, the distortion $d(\mathbf{x}, \hat{\mathbf{x}}_i)$ should be computed, then the i which minimizes that distortion should be chosen. Letting the encoder be represented by the function

$$i = Q(\mathbf{x}), \tag{3.1}$$

a formal statement of this condition is:

$$Q(\mathbf{x}) = \operatorname*{argmin}_j d(\mathbf{x}, \hat{\mathbf{x}}_j). \tag{3.2}$$

Since this means choosing to decode to the $\hat{\mathbf{x}}_i$ which is the nearest neighbor (in the minimum distortion sense) to $\mathbf{x}$, this is known as the nearest neighbor condition.

For a given encoder, the decoder is optimal if each possible reconstruction $\hat{\mathbf{x}}_i$ is optimal given the statistics of the values of $\mathbf{x}$ such that the encoder chooses to encode to that index i. In other words, for each i, $\hat{\mathbf{x}}_i$ minimizes the expected distortion given the encoder:

$$\hat{\mathbf{x}}_i = \operatorname*{argmin}_{\mathbf{y}} E[d(\mathbf{x}, \mathbf{y}) | Q(\mathbf{x}) = i]. \tag{3.3}$$

Because this optimal value of $\hat{\mathbf{x}}_i$ is the "centroid" (in the minimum distortion sense) of the region in $\mathbf{x}$ which encodes to i, this is known as the centroid condition.

These two conditions suggest one way to train the codebook. First, there must be an initial codebook (for example, one which is generated randomly). A training set of $\mathbf{x}$ values is encoded using that codebook and the nearest neighbor condition. Then, using that encoding for the training set, a new set of centroids is calculated and this becomes the new codebook. These two steps are repeated until the codebook reaches a fixed point, which is a local optimum of this training procedure. Recall that this optimum is not necessarily globally optimum because the Lloyd optimality conditions are necessary, but not sufficient, for quantizer optimality. This training procedure is known as the Linde-Buzo-Gray algorithm.[20]

More complicated training procedures can be applied to try to avoid local optimum "traps," for example the deterministic annealing method of Ken Rose[27]. Regardless of training method, once trained the codebook can be used by a nearest-neighbor encoder.

3.2 Constrained Vector Quantization

Since the best of all the locally Lloyd-optimal solutions is the global optimum, this is the "ultimate solution" in the sense of minimizing the distortion at a given rate. However, this ignores computational complexity completely. It is only because of complexity that other methods are worth considering at all.

If $\mathbf{x}$ is of dimensionality k (has k scalar components), let the complexity to compute the distortion between $\mathbf{x}$ and $\hat{\mathbf{x}}_i$ be $C(k)$, a function of k. Then the total complexity to encode using the optimal "full-search" algorithm (searching for the nearest neighbor over the full codebook) is $NC(k)$. Note that for many distortion measures (such as MSE), $C(k)$ is proportional to k, $C(k) = Ak$, in which cases the complexity is therefore NAk.

One direct way to reduce the complexity is to break up $\mathbf{x}$ into l equally sized subvectors, then perform full-search VQ on each subvector (perhaps using only one codebook, or perhaps using l different codebooks). In order to keep the rate the same as before, each subvector is encoded to one of $N^{1/l}$ indices, therefore the complexity for encoding each subvector is $N^{1/l}C(\frac{k}{l})$. The total complexity to encode all the subvectors is then $lN^{1/l}C(\frac{k}{l})$, which is $N^{1/l}Ak$ for many distortion measures. Therefore the complexity scales exponentially with vector size. This reduction in complexity is traded off with distortion.

Non-full-search methods can also be used to reduce complexity. A particular interesting class of such methods is table-lookup VQ, in which the scalar components of $\mathbf{x}$ are each uniformly quantized (say, to 8 bits) then a table indexed by that lower-precision representation of $\mathbf{x}$ contains values of indices. One especially convenient property of table-lookup VQ is that there is no computation of the distortion measure during encoding, which is advantageous for those distortion measures which would require much computation.

3.3 Finite-State Vector Quantization

Many sources to be quantized have memory, that is, there is correlation between different samples. Unlike memoryless VQ, a system which can use internal state memory about the past can take advantage of that correlation to improve performance. In such a case the vectors being coded are in a sequence.

Figure 3.2 shows a general structure for such a system, which is known as a *recursive coding system* or *coding with state-feedback* since the next state $\mathbf{s}_{t+1}$ will be fed back into the system at the next step in the sequence. In this example the vectors to be coded are a sequence $\mathbf{x}_t$ in one dimension t. In many cases of interest, t represents time.

Implicit in Figure 3.2 is that there are actually two identical "Next State Function" boxes; one is with the encoder, and one is with the decoder. Assuming that the states on both sides are initialized (both sides reset to a special agreed-upon initial state) or synchronized (the decoder's state is set to the state description as transmitted by the encoder) at some point, and assuming that there are no errors in transmitting the index i_t, both copies will run in perfect synchrony.[1] Unless there is no correlation whatsoever between successive vectors in the sequence, the state memory is expected to be able to hold information helpful to the encoding and decoding process. Since the state is known by encoder and decoder it need not be encoded for normal operation, therefore there is a savings in rate due to not sending bits for this state.

Finite-state vector quantization (FSVQ) is a recursive extension to memoryless Lloyd-optimal VQ. In FSVQ the state is a discrete index from 1 to N_s which selects

[1]In practice, errors in transmitting i_t necessitate periodic, although hopefully very infrequent, reinitialization or resynchronization of the state.

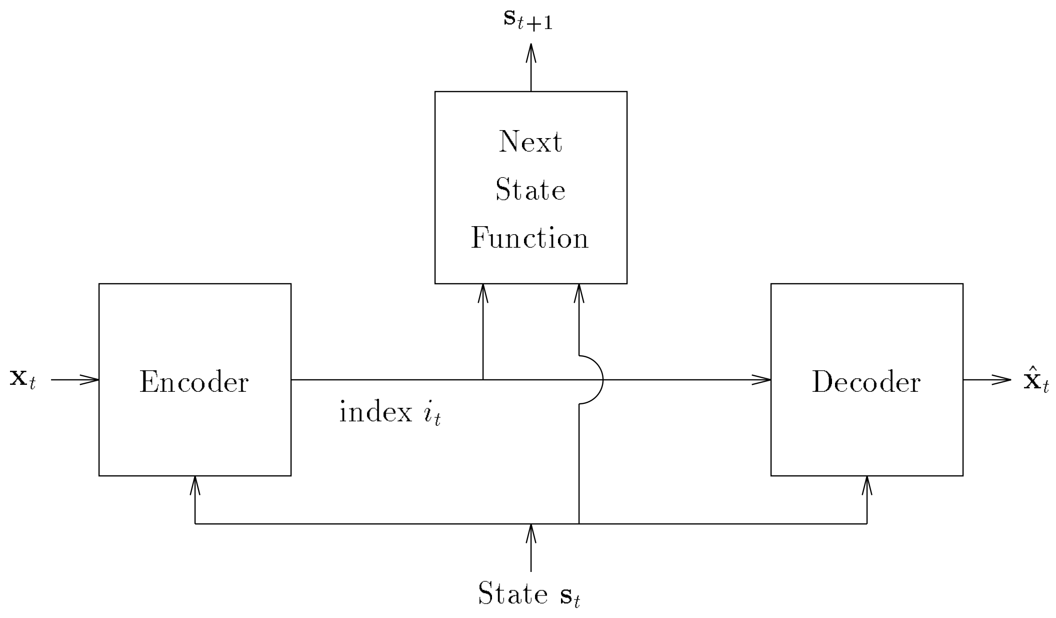

Figure 3.2: VQ with State Feedback

one codebook from a set of N_s codebooks. That codebook is then used by the encoder and decoder in the usual Lloyd-optimal way. FSVQ is also known as "switched VQ" because the value of state switches between the different codebooks.

Unfortunately, a rule for an optimal next-state function is unknown. Different types of next-state functions[2] have been tried[8, 6] and various *ad hoc* methods have been used to train the next-state function and the codebooks. However there are two prevailing problems with FSVQ.

The first problem with FSVQ is that even the best training methods lead to a design that greedily chooses a next state which may optimize the distortion at the next point in the sequence, but turns out to be a bad decision for the long-term distortion. In order to optimize the total future error, the best next state is usually not the one which minimizes the immediate short-term distortion.

A more fundamental problem is the premise of FSVQ, the restriction to finite

[2]It is interesting in the context of this thesis that Dunham's improved next-state function in [6] is a stochastic method.

states. Since normally the state will not be transmitted, there is no need to constrain it to a finite number of discrete states, which merely limits the amount of information the state can hold. However in the finite-state system, each state has an independent codebook. Therefore as the number of states increases, the number of independent parameters also increases, inevitably leading to overfitting of the data used to train those codebooks. This inherent limit to performance can only be overcome by considering a different way to represent and use the state information.

3.4 Quality Measures for Source Coding

One of the objectives of data compression is to maximize the quality of the reproduction, or stated conversely, to minimize the distortion $d(\mathbf{x}, \hat{\mathbf{x}})$ between the original data $\mathbf{x}$ and the reproduction $\hat{\mathbf{x}}$. The choice of function $d(\mathbf{x}, \hat{\mathbf{x}})$ is an important issue. Ideally the distortion measure is specifically suited to the type of data and the way in which it is to be used. For example, if the data is a still image which is to be viewed by a person, then the best distortion measures will be those for which a decrease in distortion always corresponds to an improvement in the subjective "quality" of the image as judged by that viewer. The criteria used by the viewer is different for an image like a blackboard (comprehension of handwritten words and drawings) than for a identification photo (matching the photo with a person's face) or for a reproduction of a fine art (one might wish to see even minute details like brush strokes). Likewise the criteria for telephonic speech (intelligibility of the words) is different than for an audiophile recording of a symphony orchestra (high quality reproduction of all the sounds). Therefore good distortion measures will be good mathematical models of human perception for a specific task, a topic studied in the field of psychophysics (or psychoacoustics for sound). The more perceptually accurate a distortion measure, the better one could expect a coding system (whose objective is to minimize that distortion measure) to work. The ultimate test of quality is subjective rating by humans.

A more accurate distortion measure might be rejected in favor of a less accurate one in order to reduce complexity or simplify analysis. It is for these reasons that mean squared error (MSE) is often used as a default distortion measure.

A simple example can show how perceptually unsuitable MSE is for images. An original image is modified by translating all the pixels to the right or down by a small distance, such as a shift of one or two pixels. The modified image appears to have a significant MSE when compared to the original, yet perceptually it is identical except at the very edges.

A similar example for sound waveforms again shows the unsuitability of MSE. If an original waveform is delayed by one or two samples, the MSE between original and modified waveforms is significant yet they are perceptually indistinguishable. Furthermore, if the original sound is modified simply by inverting its waveform, perceptually it is identical to the original despite a significant difference as measured by MSE.

For sound, perceptual experiments show that the effect of distortion at different frequencies is dependent on the original sound (so-called masking effects). Also, phase does not seem to be always important in perception, as the waveform-inversion example above shows.

3.5 Dithered Quantization

Perceptual experiments show that the least bothersome kind of distortion is the addition of random noise to the original data, that is, noise which is both white and uncorrelated to the original data. For example in scalar quantization of sound waveforms at a low rate, the error from a plain VQ system might be heard as degrading "fuzz" distortion, while the error from a quantization system with dithering might be perceived as a "clean" signal and white noise heard together.[16] As this is a perceptual effect, ideally the distortion measure could model this effect by rewarding noise which is less correlated. In the absence of such a distortion measure, a good objective is to force the noise to be white and uncorrelated with the original data, while increasing the nominal distortion as little as possible.

For scalar quantization with uniform reproduction values, (that is, reproduction values equally spaced by Δ) the technique of *dithering* may be used to force the desirable properties on the quantization noise. There are two basic types of dithering for uniform scalar quantization.

Subtractive dither is the addition of a "dither" signal before quantization, then the subtraction of that same signal after quantization. This can be accomplished by using a pseudo-random number generator to synchronize the dither signal at both ends of the quantization. If the dither signal meets certain conditions, then the noise will have the desired properties. An example of such a dither signal is independent identically distributed noise with a uniform distribution over $(-\Delta/2, \Delta/2]$

Non-subtractive dither is the addition of a dither signal before quantization, without the later subtraction of the signal. This avoids the complexity of synchronized, statistically well-behaved pseudo-random number generators. If the dither signal meets certain conditions which are more restrictive than in the subtractive case, then the noise will be white and a certain number of its conditional moments will not depend on the original signal.[13] An example of a dither signal for the noise power (the second moment) to be independent is independent identically distributed noise with a triangular distribution (obtained by adding two independent uniform dither signals, as described above).

3.6 Previous Research on Vector Quantization Using Neural Networks

Of previous research approaching vector quantization with ideas from the neural network field, three methods in particular are of interest in the context of this thesis. These are Teuvo Kohonen's learning vector quantization, Brian Flachs' sparse adaptive memory, and the codebook-excited neural networks work of Wu and Fallside.

3.6.1 Learning Vector Quantization

From a VQ perspective, the learning vector quantization (LVQ) work of Kohonen[3] [17] may be seen as an alternate training procedure for ordinary codebook VQ. Originally LVQ was motivated by biological models of neural activity in the brain, hence its association with "neural networks." However it does not bear much resemblance to the feedforward multi-layer neural network models used in this thesis.

The codebook vectors are associated in a one-dimensional "chain," a two-dimensional "mesh," a three-dimensional "scaffold," or so on. For example, in the one-dimensional case, each index i is considered to have two immediate "neighbors" at indices $i-1$ and $i+1$, except for the indices at the ends which have only one neighbor. In the two-dimensional case, code vectors are indexed by two indices (i,j) and are considered to have four immediate neighbors at $(i-1,j)$, $(i,j-1)$, $(i+1,j)$, and $(i,j+1)$, except for code vectors at the edges of the mesh ($i=1$ or N_i, or $j=1$ or N_j).

The concept of neighbors is used during training. For each training iteration, a training vector is presented (for example, by being drawn randomly from a training set), then its nearest neighbor code vector is found. This code vector is moved closer to the training vector by some update rule. Then its immediate neighbors on the mesh (or chain, etc.) are also moved closer to the training vector, although by a smaller amount than the nearest neighbor. As a result of this heuristic training procedure, the mesh generally will converge to an "unfolded" state in which it attempts to cover the space of probable $\mathbf{x}$.

However, from a VQ standpoint, this is merely a suboptimal heuristic training procedure. This training procedure does not promise to lead to the Lloyd-optimal solution, or even one that is nearly Lloyd-optimal, since the optimality conditions are not even taken into account by LVQ. When LVQ is able to find a better codebook than the simple Linde-Buzo-Gray algorithm it is usually a result of local minimum "traps," which more clever algorithms which do seek Lloyd optimality can overcome

[3]This method is alternately referred to in the literature as the self-organizing feature map (SOFM).

(for example, [27]). However, LVQ could be used for initial training (to get past local minima), then the result of LVQ training could be used as the starting codebook for training to optimality by the Linde-Buzo-Gray algorithm.

Wu and Fallside[35] have considered a training rule like Kohonen training without the concept of neighbors, which they call *connectionist VQ*. During training, a parameter determines how far to move the nearest neighbor toward the training vector, as in usual LVQ. For connectionist VQ, Wu and Fallside determined the value of this parameter which maximizes the immediate decrease in distortion for the current training iteration; their claim for this greedy scheme was that the distortion is the same as that obtained by training with the Linde-Buzo-Gray algorithm.

3.6.2 Sparse Adaptive Memory

The *sparse adaptive memory* (SAM) work of Brian Flachs [7] is an endeavor to apply MLP-type feedforward neural network ideas to vector quantization. The trainable parameters in SAM are a set of N "prototype" vectors $\mathbf{p}_i$ and a "codebook" matrix $\mathbf{C}$ consisting of N code vectors as its columns; however this "codebook" will not be used in the usual Lloyd VQ way. The basic SAM algorithm begins by finding the distortion $d(\mathbf{x}, \mathbf{p}_i)$ between input $\mathbf{x}$ and each of the prototype vectors $\mathbf{p}_i$. Then the minimum distortion

$$d_{\min} = \min_i d(\mathbf{x}, \mathbf{p}_i) \tag{3.4}$$

is found, from which the N "activations" a_i may be computed as

$$a_i = \begin{cases} 1 & \text{if } d(\mathbf{x}, \mathbf{p}_i) = d_{\min}, \\ 1 - \frac{1}{T}[d(\mathbf{x}, \mathbf{p}_i) - d_{\min}] & \text{if } d_{\min} < d(\mathbf{x}, \mathbf{p}_i) < d_{\min} + T \\ 0 & \text{if } d(\mathbf{x}, \mathbf{p}_i) \geq d_{\min} + T. \end{cases} \tag{3.5}$$

The critical parameter T, a positive number, is known as the transition width. Once the activations are computed, the reconstruction of $\mathbf{x}$ is found using the activations and the codebook as a thresholded linear product

$$\hat{\mathbf{x}} = \mathbf{thr}(\mathbf{Ca}), \tag{3.6}$$

where the **thr**() is the component-wise application of some continuous and differentiable threshold function such as the sigmoid or hyperbolic tangent, and $\mathbf{a}$ is the collection of all a_i into a vector. The primary innovation of SAM is that since the entire system is a continuous and differentiable function, the distortion can be minimized by gradient descent, and Flachs even calls such training backpropagation.

Inspection of Equation 3.5 reveals that there will always be at least one i for which $a_i = 1$. As long as SAM never encounters a value of $\mathbf{x}$ for which more than one a_i value is nonzero — a condition which could be guaranteed[4] by setting the transition width T to zero — then SAM can be used as an encoder and decoder, where the index is the one value of i for which $a_i = 1$ (all other $a_i = 0$). In fact plain VQ is a special case of SAM for $T = 0$, $\mathbf{p}_i = \mathbf{C}_i$ for all i, and $\mathrm{thr}(z) = z$.

However, in order for any useful training to take place, some vectors $\mathbf{x}$ must cause values of a_i to fall between 0 and 1. This is because the function of Equation 3.5 only has a non-zero derivative for the intermediate region (the second case), which only occurs for a non-zero value of T. Only with non-zero derivatives of Equation 3.5 can a non-zero gradient be propagated back to the prototype vectors to change them. That is to say, prototype training is only due to vectors which fall in the "transition region." Clearly then T must be non-zero for useful training to occur.

Unfortunately, the requirement for training that $T > 0$ conflicts with the requirement for coding that $T = 0$. Only through an arbitrary heuristic may this fundamental inconsistency be resolved. For instance, T might be started at a non-zero value for training, then over the course of training be lowered gradually to zero using a heuristic annealing schedule. No theoretical justifications for such heuristics are known.

Another unanswered question in SAM is what the output threshold function thr() should be. Again there is a conflict between the desire to use "no" threshold function $\mathrm{thr}(z) = z$, to allow SAM to act as an ordinary codebook VQ system, and the desire to choose a nonlinear threshold function such as a sigmoid, to allow more powerful

[4]This is assuming no pathological cases of $\mathbf{x}$ lying exactly on boundaries of nearest-neighbor regions, that is, assuming $d(\mathbf{x}, \mathbf{p}_i) = d_{\mathrm{min}}$ for only one value of i. If $\mathbf{x}$ is generated by a continuous probability distribution, then this will hold true because the boundaries occupy no volume of $\mathbf{x}$.

functional modeling.

Although not explored in Flachs' thesis, SAM could be extended to use state-feedback with little difficulty.

3.6.3 Codebook-Excited Neural Networks

Wu and Fallside[36] proposed a method like code-excited linear prediction (CELP) using a neural network in place of the linear prediction filter. In ordinary CELP, an excitation signal is passed through a linear prediction filter (whose parameters are somehow coded) in order to obtain the final output, which is then used to determine the distortion. A codebook of possible excitations is searched for the one which minimizes the distortion of the filter output. In Wu and Fallside's work, known as *codebook-excited neural networks* (CENN), the linear filter is simply replaced by a nonlinear filter implemented as a neural network. Without the constraint that the filter be linear, distortion can potentially be improved. However, complexity (a major problems of practical CELP implementations) is only worsened. This is because CELP may take advantage of the linearity of the filter to use shortcuts when computing the distortion for a given excitation in the codebook; these shortcuts are no longer possible with a nonlinear filter. Therefore distortion must be computed by a complete filtering operation for each excitation in the codebook.

Chapter 4

Stochastic Vector Quantization

4.1 Stochastic Vector Quantization Algorithm

The system introduced in this chapter performs the same task as plain memoryless VQ, that is, it generates a representation of a continuous-valued vector using a discrete index. As in plain VQ, once an index is generated by the encoder, the decoder simply outputs the vector found at the indexed location in the codebook table. However the encoder uses an entirely different method to generate that index.

Specifically, as illustrated in Figure 4.1, the encoder computes a probability distribution $\mathbf{p}$ over indices 1 through N using the formula

$$\begin{aligned}\mathbf{p} &= \mathbf{f}_{\mathrm{cl}}(\mathbf{x}) \\ &= \mathbf{f}_{\mathrm{normexp}}\big(\mathbf{b}_{\mathrm{u}} + \mathbf{W}_{\mathrm{u}}\mathbf{f}_{\mathrm{tanh}}(\mathbf{b}_{\mathrm{l}} + \mathbf{W}_{\mathrm{l}}\mathbf{x})\big), \text{ where}\end{aligned} \tag{4.1}$$

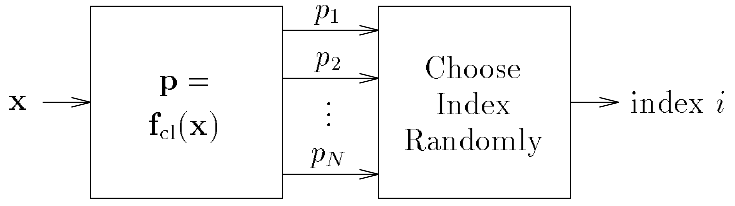

Figure 4.1: Encoder for Stochastic VQ System

$$\mathbf{f}_{\text{normexp}}(\mathbf{y}) = \frac{1}{\sum_{i=1}^{N} \exp(y_i)} \begin{pmatrix} \exp(y_1) \\ \exp(y_2) \\ \vdots \\ \exp(y_N) \end{pmatrix} \tag{4.2}$$

and where $\mathbf{f}_{\text{tanh}}(\mathbf{z})$ is the hyperbolic tangent function tanh() applied component-wise to each element of $\mathbf{z}$. As mentioned in Section 2.3.3, the function $\mathbf{f}_{\text{cl}}(\mathbf{x})$ is of the type used for multinomial classification problems in the neural network field[3]. The parameters $\mathbf{b}_{\text{u}}$, $\mathbf{b}_{\text{l}}$, $\mathbf{W}_{\text{u}}$, and $\mathbf{W}_{\text{l}}$ are the trainable weights of the system. The subscripts "u" and "l" refer to the upper layer and lower layer. Note that if the weights and $\mathbf{x}$ are finite, since the tanh() function's range is finite, it can be shown that all the elements of $\mathbf{p}$ are limited to the range $(0, 1)$. In addition, the elements of $\mathbf{p}$ sum to 1. These properties allow $\mathbf{p}$ to be interpreted as a probability distribution.

The distribution $\mathbf{p}$ is used by a random index generator to choose an index i, that is,

$$\Pr[Q(\mathbf{x}) = i|x] = \mathbf{p}_i = \mathbf{f}_{\text{cl},i}(\mathbf{x}) \tag{4.3}$$

Then $\hat{\mathbf{x}}$, the quantized representation of $\mathbf{x}$, is generated by looking up the ith vector $\hat{\mathbf{x}}_i$ in the codebook

$$\mathbf{C} = \begin{pmatrix} \hat{\mathbf{x}}_1 & \hat{\mathbf{x}}_2 & \ldots & \hat{\mathbf{x}}_N \end{pmatrix}. \tag{4.4}$$

Since the behavior of the entire system is stochastic (unlike plain VQ), the reproduction error for a given vector is an expected value over the random choice of index.

The use of randomness in quantization is of course not a new idea. This stochastic VQ scheme might be thought of as a kind of "vector dithering." Both traditional dithering and this method use a random choice of index when encoding.

One of the immediately obvious benefits of this method is that no computation of distortion measures is required during encoding, unlike usual nearest-neighbor VQ. As in table-lookup VQ, this can contribute to a significant reduction in complexity, especially when the distortion measure is computationally complex.

4.2 Decoder Optimality

If a training method can train the entire system to optimality, then since there are no constraints on the decoder's codebook, the code vectors $\hat{\mathbf{x}}_i$ of an optimally trained system will be optimal for the given encoder. In other words, the decoder would satisfy the centroid condition,

$$\hat{\mathbf{x}}_i = \operatorname*{argmin}_{\mathbf{y}} E[d(\mathbf{x}, \mathbf{y})|Q(\mathbf{x}) = i]. \tag{4.5}$$

However, it is interesting to note that the decoder can be forced to always fulfill the centroid condition. Rather than allowing the code vectors $\hat{\mathbf{x}}_i$ to take on independent values, the values can be derived from the encoder and from statistics of $\mathbf{x}$ so as to always satisfy, for all i,

$$\begin{aligned} \hat{\mathbf{x}}_i &= \operatorname*{argmin}_{\mathbf{y}} E[d(\mathbf{x}, \mathbf{y})|Q(\mathbf{x}) = i] && (4.6) \\ &= \operatorname*{argmin}_{\mathbf{y}} \int d(\mathbf{x}, \mathbf{y}) \Pr[Q(\mathbf{x}) = i|x] \Pr[\mathbf{x}]\, dx && (4.7) \\ &= \operatorname*{argmin}_{\mathbf{y}} \int d(\mathbf{x}, \mathbf{y}) \mathbf{f}_{\mathrm{cl},i}(\mathbf{x}) \Pr[\mathbf{x}]\, dx. && (4.8) \end{aligned}$$

If one wishes to satisfy the centroid condition for a set of M training vectors with respective prior probabilities w_m, the code vectors are found as

$$\hat{\mathbf{x}}_i = \operatorname*{argmin}_{\mathbf{y}} \sum_{m=1}^{M} d(\mathbf{x}_m, \mathbf{y}) \mathbf{f}_{\mathrm{cl},i}(\mathbf{x}_m) w_m \tag{4.9}$$

which is the centroid over a weighted data set, where the weighting is $\mathbf{f}_{\mathrm{cl},i}(\mathbf{x}_m) w_m$.

Another constraint which can be easily enforced is maximizing the entropy of the encoder, if desired.

4.3 Encoder Suboptimality

The decoder codebook could satisfy the centroid condition for the encoder. However, it can be shown that this encoder structure can never satisfy the nearest neighbor condition exactly. Lloyd-optimality would require the distribution $\mathbf{p}$ to be deterministic[1]

[1]This assumes a deterministic distortion function; see Section 3.5.

for a single index over each Voronoi region of the domain of $\mathbf{x}$. That is, all the elements of $\mathbf{p}$ would be exactly 0, except for a single element which would be exactly 1. But two facts about $\mathbf{f}_{\mathrm{cl}}(\mathbf{x})$ clearly conflict with this requirement: the elements of $\mathbf{p}$ can approach 0 or 1 but can never exactly equal either value, and the Lloyd-optimal discontinuous transition in the value of $\mathbf{p}$ at the boundaries between Voronoi regions can only be approximated by the continuous function $\mathbf{f}_{\mathrm{cl}}(\mathbf{x})$. For example, if the distortion function is mean squared error, the Voronoi regions are convex polytopes; the angular junctions of 3 or more polytopes are difficult to approximate well with an $\mathbf{f}_{\mathrm{cl}}(\mathbf{x})$ with a small number of parameters. It is notable however that for the special case $N = 2$ there are no such junctions, and the only region boundary is a single hyperplane which can be asymptotically achieved with a minimal-parameter $\mathbf{f}_{\mathrm{cl}}(\mathbf{x})$.

An example with $N = 3$ and $\mathbf{x} = \{\mathbf{x}_1, \mathbf{x}_2\}$ serves to illustrate this point. Suppose that the centroids of the Lloyd-optimal codebook are

$$\mathbf{C}_1 = \{1, 0\} \tag{4.10}$$

$$\mathbf{C}_2 = \{-\frac{1}{2}, \frac{\sqrt{3}}{2}\} \tag{4.11}$$

$$\mathbf{C}_3 = \{-\frac{1}{2}, -\frac{\sqrt{3}}{2}\}. \tag{4.12}$$

Then, referring to Equation 4.1, suppose the encoder's neural network has 3 "hidden units" (that is, there are 3 scalar components of $\mathbf{b}_{\mathrm{l}}$) and the weight parameters are:

$$\mathbf{W}_{\mathrm{u}} = S_u \mathbf{I} \tag{4.13}$$

$$\mathbf{b}_{\mathrm{u}} = \mathbf{0} \tag{4.14}$$

$$\mathbf{W}_{\mathrm{l}} = S_l \mathbf{C} \tag{4.15}$$

$$\mathbf{b}_{\mathrm{l}} = \mathbf{0} \tag{4.16}$$

where $\mathbf{I}$ is the identity matrix, and $\mathbf{C}$ is the Lloyd-optimal codebook as described above. The parameters S_u and S_l control the smoothness of the approximation.

Figure 4.2 shows the neural network's output for $i = 2$ in detail at the origin, for $S_u = 200$ and $S_l = 1$. Lloyd-optimal VQ would divide up the space into three identical "pie slices" meeting at the origin, which would make an equivalent figure have immediate vertically dropping "walls" at the boundaries. As Figure 4.2 shows, the neural network, as a continuous function, can only approximate those sheer drops smoothly, although as S_u or S_l increases the approximation becomes better. Figures 4.3 and 4.4 show a comparison of the Lloyd-optimal value of $\mathbf{p}_2$ (the gray line) and the neural network's value of $\mathbf{p}_2$ (the black line) at two parallel slices through Figure 4.2 which pass very near the junction of the three regions. Figure 4.3 shows $\mathbf{p}_2$ for the slice

$$\mathbf{x} = \{\mathbf{x}_1, \mathbf{x}_2\} = 0.005\{\frac{1}{2}, -\frac{\sqrt{3}}{2}\} + t\{\frac{\sqrt{3}}{2}, \frac{1}{2}\} \tag{4.17}$$

(for which the Lloyd-optimal value of $\mathbf{p}_2$ is 0) and Figure 4.4 shows $\mathbf{p}_2$ for the slice

$$\mathbf{x} = \{\mathbf{x}_1, \mathbf{x}_2\} = -0.005\{\frac{1}{2}, -\frac{\sqrt{3}}{2}\} + t\{\frac{\sqrt{3}}{2}, \frac{1}{2}\}. \tag{4.18}$$

By itself this suboptimal scheme may be useful only because of the reduced complexity during encoding, and even then should be compared against similarly suboptimal reduced-complexity methods such as table-lookup VQ. However the state-feedback extension of this scheme (discussed later in Chapter 5) can outperform finite-state VQ (the usual state-feedback extension of Lloyd-optimal VQ) by addressing some of the problems with FSVQ.

4.4 Gradient Descent Training

The encoder's weight parameters and the decoder's codebook vectors are system parameters which need to be trained. For the method of this section, only distortion measures which are continuous and differentiable functions of $\hat{\mathbf{x}}$ are allowed; however this includes nearly all useful distortion measures. The total distortion to be minimized over a training set of vectors $\mathbf{x}$ is then a continuous and differentiable function of all the parameters. This allows the use of an iterative gradient descent algorithm to seek a set of parameters which minimize distortion.

The gradient over the parameters must be found. First consider the distortion ε

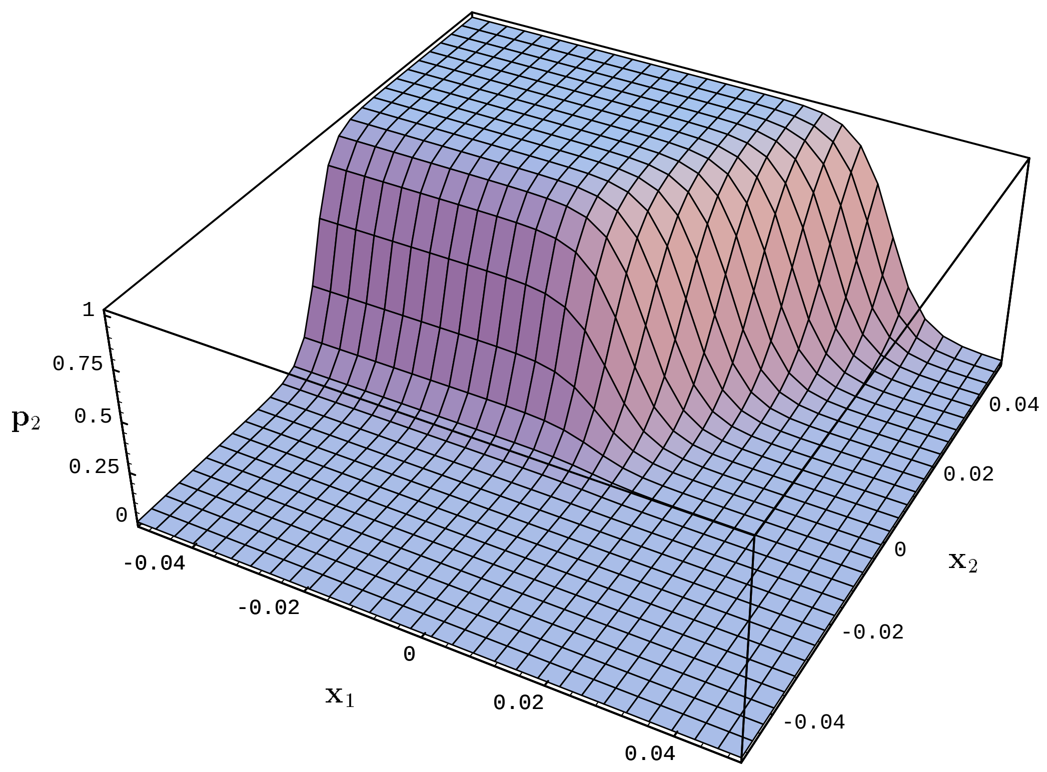

Figure 4.2: Example Encoder Distribution

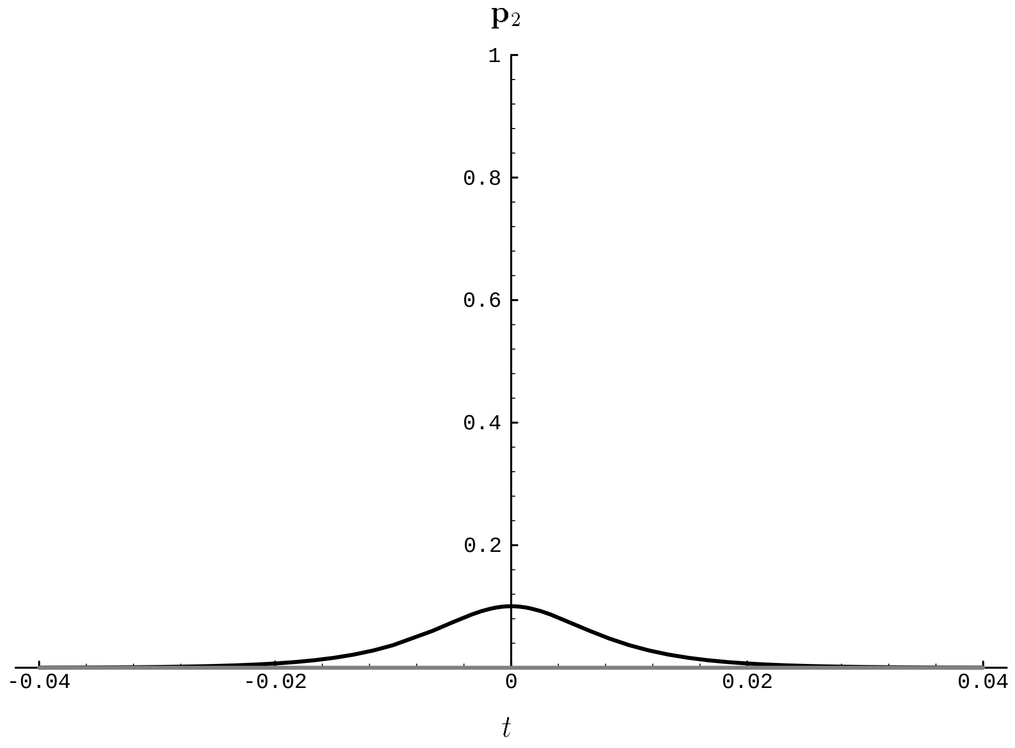

Figure 4.3: One Slice Through the Example Encoder Distribution of Figure 4.2

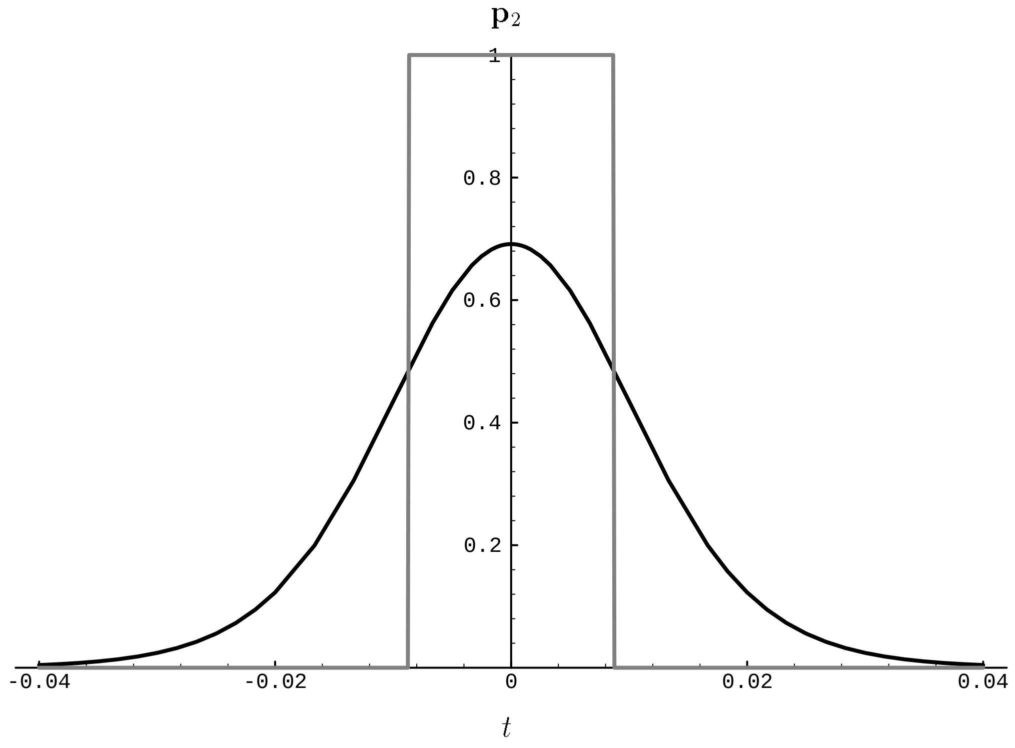

Figure 4.4: Another Slice Through the Example Encoder Distribution of Figure 4.2

for a single training set vector $\mathbf{x}$, which is computed as an expected value

$$\varepsilon = \mathrm{E}[d(\mathbf{x}, \hat{\mathbf{x}})] = \sum_{i=1}^{N} p_i d(\mathbf{x}, \hat{\mathbf{x}}_i) \tag{4.19}$$

where $d(\mathbf{x}, \hat{\mathbf{x}}_i)$ is the given distortion function (for example, MSE). Therefore

$$\frac{\partial \varepsilon}{\partial \hat{\mathbf{x}}_i} = p_i \frac{\partial d(\mathbf{x}, \hat{\mathbf{x}}_i)}{\partial \hat{\mathbf{x}}_i}, \qquad \forall i \tag{4.20}$$

and

$$\frac{\partial \varepsilon}{\partial p_i} = d(\mathbf{x}, \hat{\mathbf{x}}_i), \qquad \forall i. \tag{4.21}$$

Equation 4.20 gives the gradient for the codebook parameters, and Equation 4.21 can be used to find the gradient of the encoder's weight parameters. From $\frac{\partial \varepsilon}{\partial p_i}$ the chain rule of differentiation may be used to find the gradient of ε at the inputs of the $\mathbf{f}_{\mathrm{normexp}}()$ function. This gradient in turn can be used to find $\frac{\partial \varepsilon}{\partial \mathbf{b}_{\mathrm{u}}}$, $\frac{\partial \varepsilon}{\partial \mathbf{W}_{\mathrm{u}}}$, and the gradient at the outputs of the $\mathbf{f}_{\mathrm{tanh}}()$ function. Again the chain rule is used to find the gradient at the inputs of the $\mathbf{f}_{\mathrm{tanh}}()$ function, from which $\frac{\partial \varepsilon}{\partial \mathbf{b}_{\mathrm{l}}}$, $\frac{\partial \varepsilon}{\partial \mathbf{W}_{\mathrm{l}}}$, and $\frac{\partial \varepsilon}{\partial \mathbf{x}}$ (if desired) can be found. This method of repeatedly applying the chain rule to propagate the computation of gradient further back (toward the ultimate input of the function) is a kind of backpropagation. After backpropagation the gradient over all parameters is known.

4.5 Stochastic Gradient Descent Training

Note that the computation of the gradient in Section 4.4 required finding $\frac{\partial d(\mathbf{x}, \hat{\mathbf{x}}_i)}{\partial \hat{\mathbf{x}}_i}$ and $d(\mathbf{x}, \hat{\mathbf{x}}_i)$ for all N values of i. Suppose that, for some reason, the computation of these values is in fact very expensive for each value of i. Just such a situation is encountered in Section 5.2.

Consider an alternate method for computing not the exact gradient, but an approximation which equals the gradient in expectation. This will be done by stochastic sampling of a small number of values of i, reducing the expense of computing for all values of i.

To compute this approximation to the gradient, for each trial choose a random value j of the index using the distribution $q_1, q_2, \ldots q_N$. Then Equation 4.19 can be expressed as an expected value over many such trials:

$$\varepsilon = \sum_{j=1}^{N} p_j d(\mathbf{x}, \hat{\mathbf{x}}_j) = \sum_{j=1}^{N} q_j \frac{p_j}{q_j} d(\mathbf{x}, \hat{\mathbf{x}}_j) = \mathrm{E}_j \left[\frac{p_j}{q_j} d(\mathbf{x}, \hat{\mathbf{x}}_j) \right] \tag{4.22}$$

where $\mathrm{E}_j\,[\,]$ denotes the expected value over j. Equations 4.20 and 4.21 can be re-expressed as

$$\begin{aligned} \frac{\partial \varepsilon}{\partial \hat{\mathbf{x}}_i} &= p_i \frac{\partial d(\mathbf{x}, \hat{\mathbf{x}}_i)}{\partial \hat{\mathbf{x}}_i} = q_i \frac{p_i}{q_i} \frac{\partial d(\mathbf{x}, \hat{\mathbf{x}}_i)}{\partial \hat{\mathbf{x}}_i}, \qquad \forall i \\ &= \mathrm{E}_j \left[\begin{cases} \frac{p_j}{q_j} \frac{\partial d(\mathbf{x}, \hat{\mathbf{x}}_j)}{\partial \hat{\mathbf{x}}_j} & \text{if } i = j, \\ \mathbf{0} & \text{if } i \neq j \end{cases} \right], \qquad \forall i, \text{ and} \end{aligned} \tag{4.23}$$

$$\begin{aligned} \frac{\partial \varepsilon}{\partial p_i} &= d(\mathbf{x}, \hat{\mathbf{x}}_i) = q_i \frac{d(\mathbf{x}, \hat{\mathbf{x}}_i)}{q_i}, \qquad \forall i \\ &= \mathrm{E}_j \left[\begin{cases} \frac{d(\mathbf{x}, \hat{\mathbf{x}}_j)}{q_j} & \text{if } i = j, \\ 0 & \text{if } i \neq j \end{cases} \right], \qquad \forall i. \end{aligned} \tag{4.24}$$

Assuming a sufficiently small learning rate, the expected values in Equations 4.22, 4.23, and 4.24 could be estimated by sample averages, over a finite number of choices of j, of the bracketed values.

Effectively, the gradient is being estimated as an unbiased estimate plus some noise. The gradient descent algorithm can tolerate the addition of a certain amount of zero-mean noise to the true gradient; in fact it can be helpful in avoiding non-global minima[15]. Two methods to reduce the amount of noise can be used, separately or in combination, although both require a trade-off in the amount of training time:

- A larger number of stochastic samples can be used to make the estimate. The noise is reduced by "averaging out." The amount of computation time per iteration is proportional to the number of samples.

- The learning rate constant can be reduced. Both the noise and the magnitude of the step taken in the true gradient direction will therefore be reduced. Since the true gradient step size is reduced, more iterations will be necessary for the same amount of training progress.

The advantage of this method becomes apparent when a single sample of j is used to estimate the expected values in Equations 4.23 and 4.24; in this case one needs to calculate the values $\frac{\partial d(\mathbf{x},\hat{\mathbf{x}}_j)}{\partial \hat{\mathbf{x}}_j}$ and $d(\mathbf{x},\hat{\mathbf{x}}_j)$ only for that single value of j. This is desirable if the calculation of those values is very expensive.

4.5.1 Choice of Sampling Distribution for Training

One important question remains regarding the choice of the distribution $\mathbf{q}$. Any choice of $\mathbf{q}$ with $q_j \neq 0$ for all j will allow estimates with finite error. It would be desirable to choose $\mathbf{q}$ in order minimize the variance of the estimates. Solution of such a minimization yields the optimal $\mathbf{q}$ as a function of all N values of $\frac{\partial d(\mathbf{x},\hat{\mathbf{x}}_j)}{\partial \hat{\mathbf{x}}_j}$ (when optimizing the estimate of $\frac{\partial \varepsilon}{\partial \hat{\mathbf{x}}_i}$), or all N values of $d(\mathbf{x},\hat{\mathbf{x}}_j)$ (when optimizing the estimate of $\frac{\partial \varepsilon}{\partial p_i}$). However, if all these values were known *a priori* then there would be no need for this stochastic estimation method in the first place. Therefore from a practical standpoint, obtaining a minimal-variance estimate is not possible.

There are at least two good reasons to choose to match the sampling during training to the sampling during usage given the current weights, in other words, choose

$$\mathbf{q} = \mathbf{p}. \tag{4.25}$$

First, there is the intuitive argument that training time should be distributed proportionately to likelihood. It makes little sense to choose a uniform distribution for $\mathbf{q}$ if $\mathbf{p}$ is strongly biased toward just a few indices, since training time would be wasted on the unlikely indices. Second, it is simpler to deal with only one distribution rather than two, for computational purposes as well as monitoring training. For example, the expected value of estimated error ε for each training iteration will be the true error ε.

Chapter 5

Stochastic Vector Quantization with State-Feedback

5.1 Stochastic VQ With State Feedback Algorithm

A recurrent extension to the stochastic VQ scheme of Chapter 4 can be trained to optimality. As the results of Section 6.1 show, this optimally trained recursive system based on a suboptimal memoryless structure can sometimes outperform a non-optimally trained recursive system based on an optimal memoryless structure (FSVQ).

Figure 5.1 shows the structure of the system. The state $\mathbf{s}_t$ is a continuous-valued vector. The encoder is like the stochastic encoder of Chapter 4 except that $\mathbf{f}_{\mathrm{cl}}()$ must be expanded to accommodate the additional input $\mathbf{s}_t$:

$$\begin{aligned}\mathbf{p}_t &= \mathbf{f}_{\mathrm{cl}}(\mathbf{x}_t, \mathbf{s}_t) \\ &= \mathbf{f}_{\mathrm{normexp}}\big(\mathbf{b}_{\mathrm{u}} + \mathbf{W}_{\mathrm{u}}\mathbf{f}_{\mathrm{tanh}}(\mathbf{b}_{\mathrm{l}} + \mathbf{W}_{\mathrm{l,x}}\mathbf{x}_t + \mathbf{W}_{\mathrm{l,enc,s}}\mathbf{s}_t)\big)\end{aligned} \tag{5.1}$$

In order to reduce complexity, the decoder and the next-state function are combined.[1] As in the memoryless case, the index i_t is used to select the vector

[1]It would be possible to have two of the dashed-line boxes of Figure 5.1, one with codebook matrix $\mathbf{C}_{\mathrm{dec}}$ feeding into $\mathbf{f}_{\mathrm{dec}}()$, and one with codebook matrix $\mathbf{C}_{\mathrm{ns}}$ feeding into $\mathbf{f}_{\mathrm{ns}}()$. However, there is no reason not to combine them in order to allow $\mathbf{f}_{\mathrm{dec}}()$ and $\mathbf{f}_{\mathrm{ns}}()$ to "share hidden units," that is, share the common term of Equation 5.4.

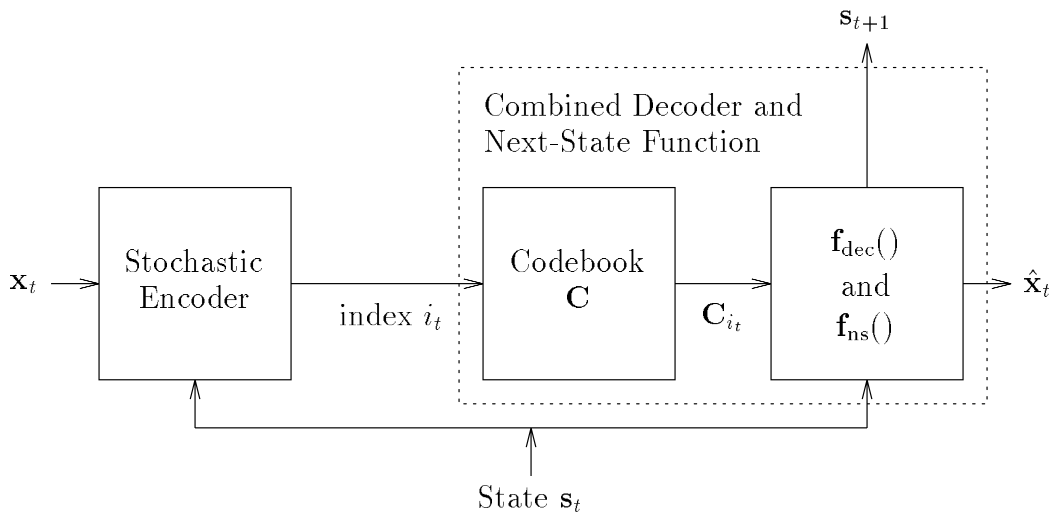

Figure 5.1: Stochastic VQ with State Feedback

$\mathbf{C}_{i_t}$ from a codebook matrix $\mathbf{C}$. Then $\hat{\mathbf{x}}_t$ and $\mathbf{s}_{t+1}$ are found by

$$\begin{aligned}\mathbf{s}_{t+1} &= \mathbf{f}_{\text{ns}}(\mathbf{C}_{i_t}, \mathbf{s}_t) \\ &= \mathbf{f}_{\text{tanh}}\big(\mathbf{b}_{\text{ns}} + \mathbf{W}_{\text{ns}}\mathbf{f}_{\text{tanh}}(\mathbf{b}_{\text{l,dec}} + \mathbf{W}_{\text{l,dec,}\mathbf{C}}\mathbf{C}_{i_t} + \mathbf{W}_{\text{l,dec,s}}\mathbf{s}_t)\big)\end{aligned} \tag{5.2}$$

and

$$\begin{aligned}\hat{\mathbf{x}}_t &= \mathbf{f}_{\text{dec}}(\mathbf{C}_{i_t}, \mathbf{s}_t) \\ &= \mathbf{b}_{\text{bias}} + \mathbf{W}_{\text{scale}}\mathbf{f}_{\text{tanh}}\big(\mathbf{b}_{\text{u,dec}} + \mathbf{W}_{\text{u,dec}}\mathbf{f}_{\text{tanh}}(\mathbf{b}_{\text{l,dec}} + \mathbf{W}_{\text{l,dec,}\mathbf{C}}\mathbf{C}_{i_t} + \mathbf{W}_{\text{l,dec,s}}\mathbf{s}_t)\big)\end{aligned} \tag{5.3}$$

where $\mathbf{W}_{\text{scale}}$ is a diagonal matrix. The terms $\mathbf{b}_{\text{bias}}$ and $\mathbf{W}_{\text{scale}}$ allow the ranges of the components of $\hat{\mathbf{x}}_t$ to be unrestricted. The argument of the common term

$$\mathbf{f}_{\text{tanh}}(\mathbf{b}_{\text{l,dec}} + \mathbf{W}_{\text{l,dec,}\mathbf{C}}\mathbf{C}_{i_t} + \mathbf{W}_{\text{l,dec,s}}\mathbf{s}_t) \tag{5.4}$$

can be reformulated by defining a new codebook $\mathbf{C}'$ such that

$$\mathbf{C}' = (\mathbf{b}_{\text{l,dec}}, \mathbf{b}_{\text{l,dec}}, \ldots \mathbf{b}_{\text{l,dec}}) + \mathbf{W}_{\text{l,dec,}\mathbf{C}}\mathbf{C}, \tag{5.5}$$

which allows the computation of merely

$$\mathbf{f}_{\text{tanh}}(\mathbf{C}'_{i_t} + \mathbf{W}_{\text{l,dec,s}}\mathbf{s}_t) \tag{5.6}$$

further reducing the implementation complexity.

5.2 Gradient Descent Training

The parameters of the system which need to be trained are:

- $\mathbf{b}_{\mathrm{u}}$, $\mathbf{W}_{\mathrm{u}}$, $\mathbf{b}_{\mathrm{l}}$, $\mathbf{W}_{\mathrm{l},\mathbf{x}}$, and $\mathbf{W}_{\mathrm{l,enc,s}}$ from Equation 5.1,
- $\mathbf{b}_{\mathrm{ns}}$ and $\mathbf{W}_{\mathrm{ns}}$ from Equation 5.2,
- $\mathbf{b}_{\mathrm{bias}}$, $\mathbf{W}_{\mathrm{scale}}$, $\mathbf{b}_{\mathrm{u,dec}}$, and $\mathbf{W}_{\mathrm{u,dec}}$ from Equation 5.3,
- $\mathbf{C}'$ and $\mathbf{W}_{\mathrm{l,dec,s}}$ from Equation 5.6, and
- an initial state vector[2] $\mathbf{s}_1$.

Let $\mathbf{w}$ represent all those system parameters except $\mathbf{s}_1$. Assuming a continuous and differentiable distortion function, since the total distortion is a continuous function of $\mathbf{w}$ and $\mathbf{s}_1$, a gradient descent algorithm can be used to train $\mathbf{w}$ and $\mathbf{s}_1$.

Consider a sequence of T vectors $\{\mathbf{x}_1, \mathbf{x}_2, \ldots \mathbf{x}_T\}$. Define the total current and future distortion at step t given this sequence, given that the current state is $\mathbf{s}_t$, and given that the system parameters are $\mathbf{w}$, as $\varepsilon_t(\mathbf{s}_t, \mathbf{w})$; that is

$$\varepsilon_t(\mathbf{s}_t, \mathbf{w}) = \sum_{i=1}^{N} \mathbf{p}_{i_t}\big(d(\mathbf{x}_t, \hat{\mathbf{x}}_t) + \varepsilon_{t+1}(\mathbf{s}_{t+1}, \mathbf{w})\big). \tag{5.7}$$

The total distortion over the entire sequence, which is the quantity to be minimized, is

$$\varepsilon = \varepsilon_1(\mathbf{s}_1, \mathbf{w}). \tag{5.8}$$

[2]The value of $\mathbf{s}_1$ may be established arbitrarily (for example $\mathbf{0}$) but the additional complexity to train the value to optimality is negligible.

The gradients $\frac{\partial \varepsilon}{\partial \mathbf{w}}$ and $\frac{\partial \varepsilon}{\partial \mathbf{s}_1}$ need to be found. Omitting some arguments for readability, differentiation of Equation 5.7 gives

$$\begin{aligned}\frac{\partial \varepsilon_t(\mathbf{s}_t,\mathbf{w})}{\partial \mathbf{w}} = \sum_{i=1}^{N} \Bigg(& \frac{\partial \mathbf{f}_{\mathrm{cl},i}(\mathbf{s}_t,\mathbf{x}_t,\mathbf{w})}{\partial \mathbf{w}} \big(d(\mathbf{x}_t,\hat{\mathbf{x}}_t) + \varepsilon_{t+1}(\mathbf{s}_{t+1},\mathbf{w})\big) \\ & + \mathbf{p}_{i_t} \Bigg(\frac{\partial \mathbf{f}_{\mathrm{dec}}(\mathbf{s}_t,i,\mathbf{w})}{\partial \mathbf{w}} \frac{\partial d(\mathbf{x}_t,\mathbf{f}_{\mathrm{dec}}())}{\partial \mathbf{f}_{\mathrm{dec}}} \\ & + \frac{\partial \mathbf{f}_{\mathrm{ns}}(\mathbf{s}_t,i,\mathbf{w})}{\partial \mathbf{w}} \frac{\partial \varepsilon_{t+1}(\mathbf{s}_{t+1},\mathbf{w})}{\partial \mathbf{s}_{t+1}} + \frac{\partial \varepsilon_{t+1}(\mathbf{s}_{t+1},\mathbf{w})}{\partial \mathbf{w}} \Bigg)\Bigg)\end{aligned} \tag{5.9}$$

and

$$\begin{aligned}\frac{\partial \varepsilon_t(\mathbf{s}_t,\mathbf{w})}{\partial \mathbf{s}_t} = \sum_{i=1}^{N} \Bigg(& \frac{\partial \mathbf{f}_{\mathrm{cl},i}(\mathbf{s}_t,\mathbf{x}_t,\mathbf{w})}{\partial \mathbf{s}_t} \big(d(\mathbf{x}_t,\hat{\mathbf{x}}_t) + \varepsilon_{t+1}(\mathbf{s}_{t+1},\mathbf{w})\big) \\ & + \mathbf{p}_{i_t} \Bigg(\frac{\partial \mathbf{f}_{\mathrm{dec}}(\mathbf{s}_t,i,\mathbf{w})}{\partial \mathbf{s}_t} \frac{\partial d(\mathbf{x}_t,\mathbf{f}_{\mathrm{dec}}())}{\partial \mathbf{f}_{\mathrm{dec}}} \\ & + \frac{\partial \mathbf{f}_{\mathrm{ns}}(\mathbf{s}_t,i,\mathbf{w})}{\partial \mathbf{s}_t} \frac{\partial \varepsilon_{t+1}(\mathbf{s}_{t+1},\mathbf{w})}{\partial \mathbf{s}_{t+1}} \Bigg)\Bigg)\end{aligned} \tag{5.10}$$

The term $\frac{\partial d(\mathbf{x}_t,\mathbf{f}_{\mathrm{dec}}())}{\partial \mathbf{f}_{\mathrm{dec}}}$ is computed as a derivative of the distortion function. The terms

- $\frac{\partial \mathbf{f}_{\mathrm{cl},i}(\mathbf{s}_t,\mathbf{x}_t,\mathbf{w})}{\partial \mathbf{w}}$ and $\frac{\partial \mathbf{f}_{\mathrm{cl},i}(\mathbf{s}_t,\mathbf{x}_t,\mathbf{w})}{\partial \mathbf{s}_t}$,
- $\frac{\partial \mathbf{f}_{\mathrm{dec}}(\mathbf{s}_t,i,\mathbf{w})}{\partial \mathbf{w}}$ and $\frac{\partial \mathbf{f}_{\mathrm{dec}}(\mathbf{s}_t,i,\mathbf{w})}{\partial \mathbf{s}_t}$, and
- $\frac{\partial \mathbf{f}_{\mathrm{ns}}(\mathbf{s}_t,i,\mathbf{w})}{\partial \mathbf{w}}$ and $\frac{\partial \mathbf{f}_{\mathrm{ns}}(\mathbf{s}_t,i,\mathbf{w})}{\partial \mathbf{s}_t}$

are computed by chain-rule differentiation (backpropagation) of their respective functions at the given input values.

Equations 5.7, 5.9, and 5.10 together define a recursive relationship which allows one to find $\varepsilon_t(\mathbf{s}_t,\mathbf{w})$, $\frac{\partial \varepsilon_t(\mathbf{s}_t,\mathbf{w})}{\partial \mathbf{w}}$, and $\frac{\partial \varepsilon_t(\mathbf{s}_t,\mathbf{w})}{\partial \mathbf{s}_t}$ from their corresponding values at step $t+1$. Note that

$$\varepsilon_{T+1}(\mathbf{s}_{T+1},\mathbf{w}) = 0, \text{ therefore} \tag{5.11}$$

$$\frac{\partial \varepsilon_{T+1}(\mathbf{s}_{T+1},\mathbf{w})}{\partial \mathbf{w}} = \mathbf{0} \text{ and} \tag{5.12}$$

$$\frac{\partial \varepsilon_{T+1}(\mathbf{s}_{T+1},\mathbf{w})}{\partial \mathbf{s}_{T+1}} = \mathbf{0}, \tag{5.13}$$

which provides initial conditions for recursion. After recursion all the way back to $t = 1$,

$$\frac{\partial \varepsilon}{\partial \mathbf{w}} = \frac{\partial \varepsilon_1(\mathbf{s}_1, \mathbf{w})}{\partial \mathbf{w}} \text{ and} \tag{5.14}$$

$$\frac{\partial \varepsilon}{\partial \mathbf{s}_1} = \frac{\partial \varepsilon_1(\mathbf{s}_1, \mathbf{w})}{\partial \mathbf{s}_1} \tag{5.15}$$

are known. This method resembles backpropagation-through-time (BPTT) and could be said to be a variant of BPTT. As usual, for multiple sequences in the training set, each sequence's gradient is calculated individually and then the gradients are averaged over the set.

5.3 Stochastic Gradient Descent Training

The recursive form of Equations 5.7, 5.9, and 5.10 implies that, when beginning with a given initial state $\mathbf{s}_1$, at step t there are N^t next states $\mathbf{s}_{t+1}$ over which terms to be summed must be calculated. This means that a total number of $\sum_{t=1}^{T} N^t$ such terms must be calculated in order to find the gradients as described in Section 5.2. Since[3] for large T,

$$\sum_{t=1}^{T} N^t \approx \frac{N^{T+1}}{N-1} = \frac{N}{N-1} N^T, \tag{5.16}$$

the growth in complexity is approximately exponential in T; therefore computation of the gradients using this method is prohibitively expensive for nontrivial values of T. This is the complexity problem alluded to in Section 4.5.

One solution to this problem is to compute an estimate of the true gradients by stochastically choosing a smaller number of indices at each t and $\mathbf{s}_t$. In the extreme, if only a single index is chosen at each step t, the number of terms to be computed is

$$\sum_{t=1}^{T} 1^t = T, \tag{5.17}$$

[3]This is assuming that $N > 1$, without which this system can only produce a single output sequence $\{\hat{\mathbf{x}}_t\}$.

certainly a more reasonable complexity than $\frac{N}{N-1}N^T$.

The development of Section 4.5 can be extended to this case. To compute these estimates, given state $\mathbf{s}_t$ at step t, for each trial choose a random value j_t of the index using distribution $q_{t,1}, q_{t,2}, \ldots, q_{t,N}$ and calculate $\hat{\mathbf{x}}_t$ and $\mathbf{s}_{t+1}$ using that index. Equations 5.7, 5.9, and 5.10 may then be re-expressed as expected values over many such trials:

$$\varepsilon_t(\mathbf{s}_t, \mathbf{w}) = \mathrm{E}_{j_t}\left[\frac{\mathbf{p}_{i_t}}{\mathbf{q}_{j_t}}\left(d(\mathbf{x}_t, \hat{\mathbf{x}}_t) + \varepsilon_{t+1}(\mathbf{s}_{t+1}, \mathbf{w})\right)\right], \tag{5.18}$$

$$\begin{aligned}\frac{\partial \varepsilon_t(\mathbf{s}_t, \mathbf{w})}{\partial \mathbf{w}} = \mathrm{E}_{j_t}\Bigg[& \frac{1}{\mathbf{q}_{j_t}} \frac{\partial \mathbf{f}_{\mathrm{cl},i}(\mathbf{s}_t, \mathbf{x}_t, \mathbf{w})}{\partial \mathbf{w}}\left(d(\mathbf{x}_t, \hat{\mathbf{x}}_t) + \varepsilon_{t+1}(\mathbf{s}_{t+1}, \mathbf{w})\right) \\ & + \frac{\mathbf{p}_{i_t}}{\mathbf{q}_{j_t}}\Bigg(\frac{\partial \mathbf{f}_{\mathrm{dec}}(\mathbf{s}_t, i, \mathbf{w})}{\partial \mathbf{w}} \frac{\partial d(\mathbf{x}_t, \mathbf{f}_{\mathrm{dec}}())}{\partial \mathbf{f}_{\mathrm{dec}}} \\ & \quad + \frac{\partial \mathbf{f}_{\mathrm{ns}}(\mathbf{s}_t, i, \mathbf{w})}{\partial \mathbf{w}} \frac{\partial \varepsilon_{t+1}(\mathbf{s}_{t+1}, \mathbf{w})}{\partial \mathbf{s}_{t+1}} + \frac{\partial \varepsilon_{t+1}(\mathbf{s}_{t+1}, \mathbf{w})}{\partial \mathbf{w}} \Bigg)\Bigg],\end{aligned} \tag{5.19}$$

and

$$\begin{aligned}\frac{\partial \varepsilon_t(\mathbf{s}_t, \mathbf{w})}{\partial \mathbf{s}_t} = \mathrm{E}_{j_t}\Bigg[& \frac{1}{\mathbf{q}_{j_t}} \frac{\partial \mathbf{f}_{\mathrm{cl},i}(\mathbf{s}_t, \mathbf{x}_t, \mathbf{w})}{\partial \mathbf{s}_t}\left(d(\mathbf{x}_t, \hat{\mathbf{x}}_t) + \varepsilon_{t+1}(\mathbf{s}_{t+1}, \mathbf{w})\right) \\ & + \frac{\mathbf{p}_{i_t}}{\mathbf{q}_{j_t}}\Bigg(\frac{\partial \mathbf{f}_{\mathrm{dec}}(\mathbf{s}_t, i, \mathbf{w})}{\partial \mathbf{s}_t} \frac{\partial d(\mathbf{x}_t, \mathbf{f}_{\mathrm{dec}}())}{\partial \mathbf{f}_{\mathrm{dec}}} \\ & \quad + \frac{\partial \mathbf{f}_{\mathrm{ns}}(\mathbf{s}_t, i, \mathbf{w})}{\partial \mathbf{s}_t} \frac{\partial \varepsilon_{t+1}(\mathbf{s}_{t+1}, \mathbf{w})}{\partial \mathbf{s}_{t+1}} \Bigg)\Bigg].\end{aligned} \tag{5.20}$$

Assuming a sufficiently small learning rate, the expected values may be estimated by the bracketed values for a single sequence of indices $\{j_1, j_2, \cdots, j_N\}$. As discussed in Section 4.5, such estimates effectively add zero-mean noise of a controllable amount (more training time can be sacrificed for less noise) to the true gradient; the gradient descent algorithm can tolerate such noise, and may even be helped by it.

The distributions $\mathbf{q}_t$ must be chosen. Any choice of $\mathbf{q}_t$ with $q_{t,j} \neq 0$ for all j and all t will allow estimates with finite error. For reasons discussed in Section 4.5.1, $\mathbf{q}_t = \mathbf{p}_t$ is chosen for each t.

5.4 Centroid Constraint

Section 4.2 introduced a constraint to insure decoder optimality by forcing the decoder outputs to be centroids for the given encoder. This constraint can be extended to the state-feedback case. However, the constraint would be seen to depend on terms for each of all N^T possible index sequences. Unfortunately, such computations are intractable in practice, as was shown for finding the exact gradient in Section 5.3. While it is reasonable to use a noisy estimate of gradient in iterative gradient descent training, enforcing this constraint inexactly by forcing a noisy estimate of the correct codebook is not a good idea. Therefore this constraint is no longer practical for the state-feedback case.

Chapter 6

Experiments

6.1 Gauss-Markov Series

For direct comparison to the FSVQ results of Foster, Gray, and Dunham[8], this stochastic system with state-feedback was trained on a Gauss-Markov source, and experimental rate-distortion performance was observed.

Specifically, the source used was a Gauss-Markov source $\{y_n\}$ defined by

$$y_{n+1} = ay_n + r_n \tag{6.1}$$

where $\{r_n\}$ is a zero-mean unit-variance independent, identically distributed Gaussian series, and $a = 0.9$. A training sequence of length $128,000$ samples and a separate test sequence of the same length were generated. For each of the sequences $\{\mathbf{x}_t\}$, a sequence of vectors of dimension 4, was made by partitioning $\{y_n\}$ into vectors of 4 consecutive samples, that is

$$\mathbf{x}_t = \begin{pmatrix} y_{4t} & y_{4t+1} & y_{4t+2} & y_{4t+3} \end{pmatrix}. \tag{6.2}$$

Since this data is artificial and is not meant to be perceived by humans, any distortion measure could be used. Therefore, for simplicity, squared difference (MSE) was used as the distortion function. This experimental setup is exactly as in [8], in order to allow direct comparison to the results of that study.

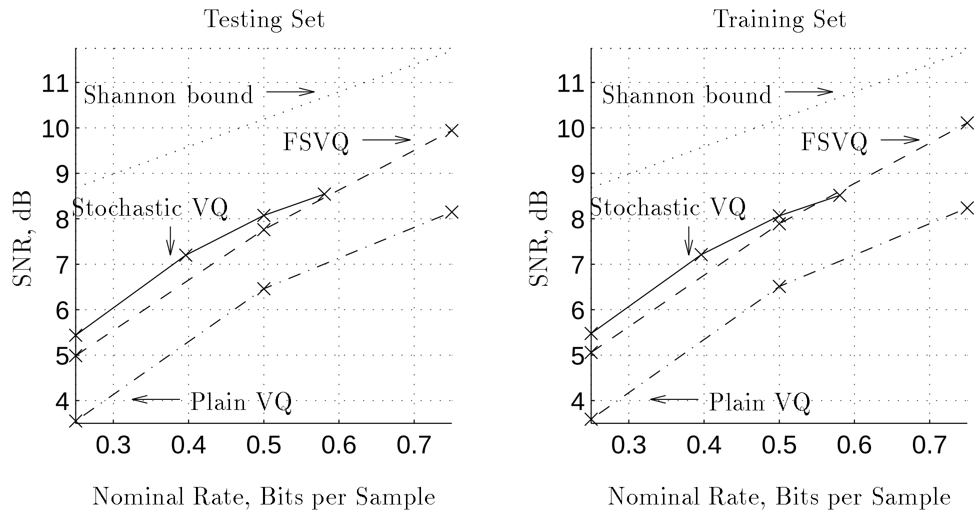

Figure 6.1: Performance of Stochastic VQ with State-Feedback, FSVQ, and Plain VQ for Gauss-Markov Data

Figure 6.1 shows a comparison of performance of stochastic VQ with state-feedback, FSVQ (with 128 states), and plain VQ on the Gauss-Markov testing and training data sets. The rate is reported as nominal rate in bits per sample ($\frac{1}{4}\log_2(N)$) rather than as entropy, in keeping with the results in [8]. The FSVQ results are for "omniscient labeled transition FSVQ," the best of 4 training methods considered in [8], and come directly from Foster et al. Also shown is the Shannon bound on rate-distortion performance, which is known since this source is generated from a known distribution.

For the testing set, which predicts the performance on novel data better than does the training set, stochastic VQ can perform up to 1.89 dB better than plain VQ and 0.45 dB better than FSVQ, although the improvements fall off at higher rates. As expected, since the source has high correlation between vectors in the sequence, both state-memory methods out-perform plain memoryless VQ by a significant margin.

It is also noteworthy that the difference between the training and testing performance ("overfitting" of the training data) is much less for stochastic VQ (less than 0.04 dB) than for FSVQ (0.13 dB or higher).

These results confirm that the training algorithm for stochastic VQ with state-feedback works well, and suggest that this scheme may be overcoming some of the difficulties of FSVQ (such as greedy next-state decisions, and the finite-state constraint). However the Shannon bound shows that there is still much room for improvement, at least for this type of source.

6.2 Speech Coding

A more practical experiment was undertaken to evaluate the application to speech coding. The stochastic VQ with state-feedback method was used directly on the speech waveform, with sets of successive samples partitioned into vector sequences.

This system was used for coding speech sampled at a rate of 8000 hertz, which is standard for telephony. Four different bit rates were used: 4000 bits per second ($N = 2$ with 2-sample length vectors), 8000 bits per second ($N = 2$, 1-sample vectors), 12679.7 bits per second ($N = 3$, 1-sample vectors),[1] and 16000 bits per second ($N = 4$, 1-sample vectors).

The results for this system were compared to the state-of-the-art 4800 bits per second code-excited linear prediction (CELP) speech coder (Federal Standard 1016) as described in Campbell, et al.[1] In the CELP method, a 240-sample "frame" of speech is analyzed to extract a 10-tap linear prediction filter; these 10 parameters are simply coded with scalar quantization. Then the algorithm searches a codebook of possible "excitation" input waveforms for the given filter. The distortion measure which is being minimized by the search is a special perceptual distortion measure; this is computed as the power of the difference signal (between the original speech waveform and the filter's output waveform for that frame) filtered by a "perceptual weighting filter" which depends on the 10 linear prediction parameters for the frame. More details about the perceptually-weighted filtering are contained in Appendix A.

[1]Note that practical implementation of the $N = 3$ rate directly on an inherently binary channel requires combining M successive symbols for a true rate of $\lceil M \log_2 3 \rceil \frac{8000}{M}$ bits per second; therefore the choice of M allows a tradeoff between minimizing wasted bits, or minimizing delay (to gather M symbols) and complexity (to pack symbols into bits and unpack symbols from bits). Fortunately the reasonable value $M = 5$ yields a rate of 12800 bits per second, a waste of only about 1 %.

The “net” bit rate, without error-correction or synchronization bits, is 4600 bits per second. This is the rate for a fair comparison with the stochastic method. An implementation of this CELP coder available via the Internet[2] programmed by the authors of [1] was used for comparison.

In order that a fair comparison of a numerical distortion measure can be made, the stochastic VQ with state-feedback is trained to minimize exactly the same perceptually-weighted-filtered distortion measure as CELP. This measure is certainly an improvement over simple MSE, but leaves much to be desired. For example, the power of a difference signal — even a filtered difference signal, as is used here — depends very much on the relative phase of the two waveforms, while simple examples (such as inverting one of the waveforms) show that phase is not of utmost importance in aural perception. This suggests that perceptually useless information is being encoded at the expense of perceptually important information. For a truly practical application to speech coding a better distortion measure should be used during training; however for a first attempt — and more importantly, for direct comparison to CELP — this measure is appropriate. In the results, quality (rather than distortion) is presented as signal-to-perceptual-noise-ratio (SPNR), the ratio of the signal power to distortion measure in decibels.

Finally, the computational complexity required to encode and decode is of great interest in speech coding. Certainly complexity is implementation-dependent, but gross differences in complexity tend to persist regardless of implementation. As mentioned, an implementation of CELP by the authors of [1] was regarded as a reasonable measure of CELP's complexity when compared to the implementation of stochastic VQ with state-feedback. Both implementations are written as fairly efficient programs, both do floating-point arithmetic, and both were run on a Sun Sparc-20 workstation. Complexity was measured by using the UNIX *time* utility program to find the CPU time. Run-time was averaged over several runs; this average time was found for speech segments of different lengths in order to find the marginal processing time per segment time (that is, factor out the start-up time which would become insignificant as the length of speech segment increases). A complexity of 1.0 on this scale indicates that the encoding (or decoding) could be done in real-time on a Sun Sparc-20.

The data used for training and testing of the stochastic VQ system (and evaluation of CELP) was included in the distribution of the CELP implementation. One speech segment of length from 2.5 seconds to 3.5 seconds from each of three male and three female speakers comprised the entire data set of six speech segments. All were native American English speakers, and the samples were recorded in variously noisy environments, not atypical for practical telephony. The resolution of 16 bits per sample was more than adequate for the telephone-quality coding to be performed on these samples. This entire data set was partitioned into the training set (two male speakers and one female) and testing set (the remaining two females and one male). The post-training quality results over the testing set, the nominal rates, and the complexity measures are compared for the stochastic VQ with state-feedback and for CELP in Figures 6.2, 6.3, and 6.4.

The most obvious advantage of the stochastic VQ over CELP is the difference in encoding complexity. In order to encode at the same bit rate, CELP is about 10 times as complex as stochastic VQ; to achieve the same quality CELP is about 8 times as complex. CELP's complexity comes mostly from the computation required to evaluate the distortion measure during the codebook search. For each codebook entry this requires an operation equivalent to filtering by the linear prediction filter, then filtering the difference filter by the perceptual weighting filter. On the other hand, during encoding the stochastic VQ method does no such evaluation of distortion, much to its advantage.

The comparison of quality to rate in Figure 6.4 shows that the CELP standard can perform at the same quality as stochastic VQ with state-feedback at about half the bit rate. Considering the sophistication of the CELP method this is an encouraging result.

Listening reveals that the CELP-encoded speech sounds rather "synthetic" or "robotic" (symptomatic of low-rate linear-predictive coding techniques) while the stochastic VQ-encoded speech sounds "natural" but noisy. As mentioned previously

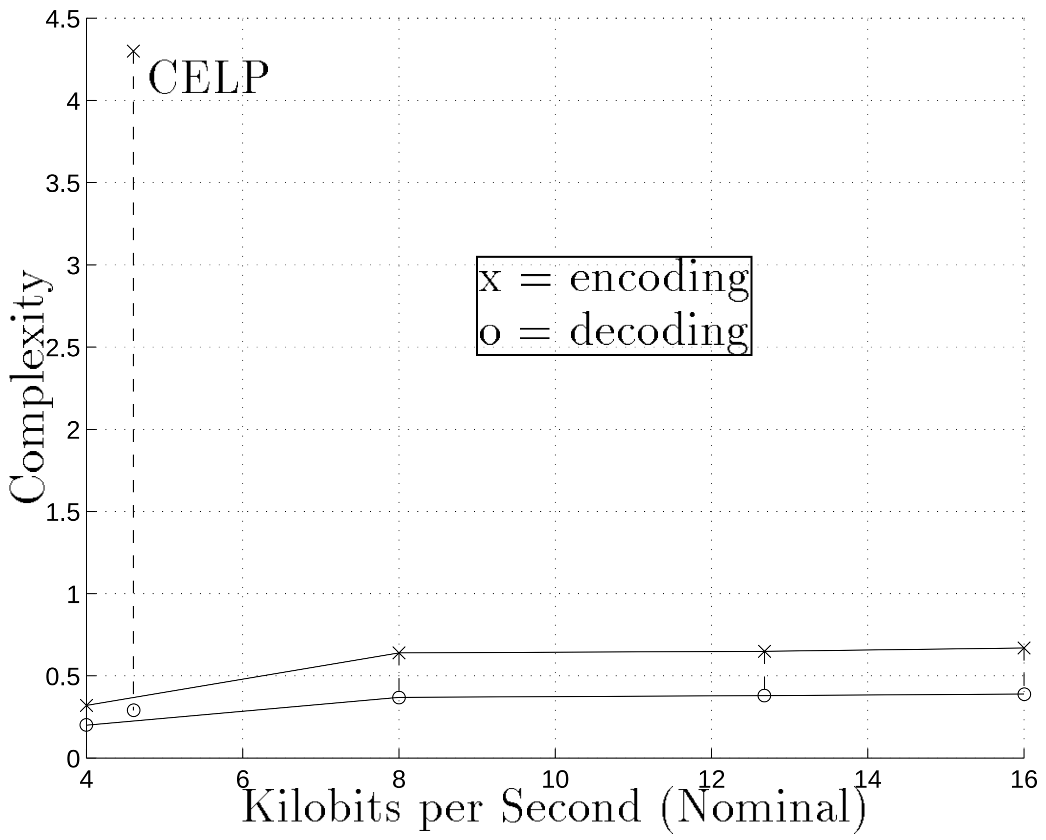

Figure 6.2: Complexity vs. Rate for Testing Set

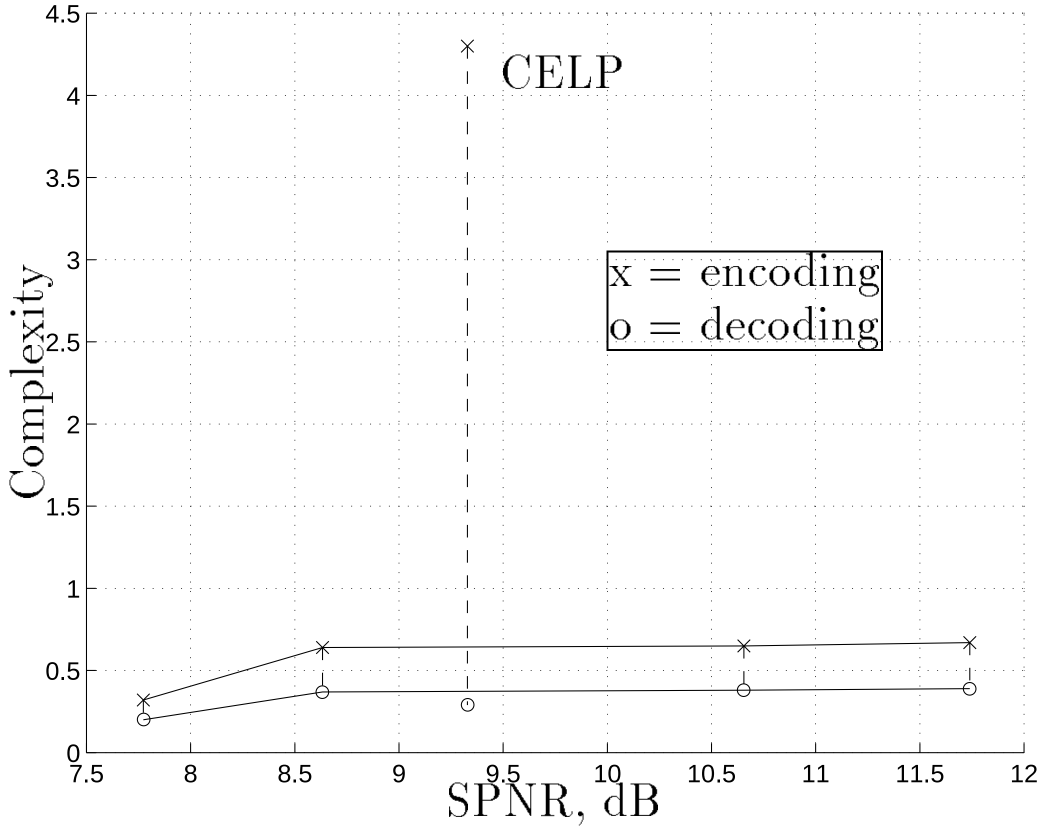

Figure 6.3: Complexity vs. Quality for Testing Set

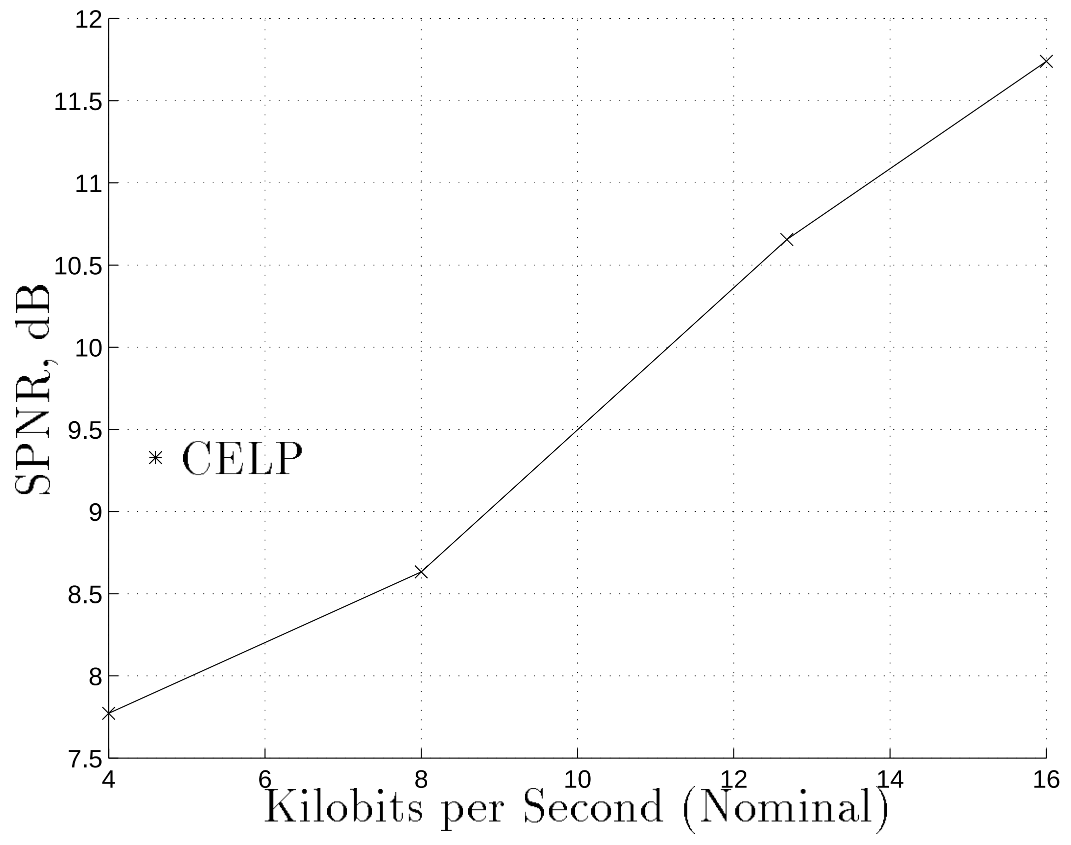

Figure 6.4: Quality vs. Rate for Testing Set

listening tests are not a completely fair comparison for this experiment since a suboptimal distortion measure was used during training. Sample speech segments in their original and variously coded versions are currently available for listening on the World Wide Web.[18]

Another point of comparison is the algorithmic delay. Since CELP must process an entire frame of 240 samples all at once, the algorithm introduces a non-trivial delay of 30 milliseconds, while the stochastic VQ method which processes vectors of one or two samples has no algorithmic delay, or an algorithmic delay of one sample (0.125 milliseconds). As mentioned in Chapter 3 low delay is desirable in telephone networks.

Chapter 7

Conclusions and Future Directions

Stochastic VQ as presented in this dissertation is a novel technique for data compression which draws upon ideas from the neural network field. Contributions of this method are:

- A small number of parameters can be used, minimizing the penalty of overfitting.
- A training procedure (gradient descent) can train the parameters to an optimum.
- During encoding, this method does not compute the distortion measures over the codebook, which is advantageous for complex distortion measures.

Stochastic VQ with state-feedback is more useful as it has those advantages, and in addition:

- The state vector is continuous, as opposed to FSVQ which may only hold a limited amount of state information in memory.
- The training procedure (gradient descent) seeks an optimum of long-term distortion, not greedy short-term distortion.

More investigation of the basic algorithm would be of interest. In particular, experimental comparison to other methods such as those discussed in Section 3.6 would be relevant. Comparison to a state-feedback extension of sparse adaptive memory, or comparison of the memoryless versions of SAM and this work, might be particularly interesting. Also, the training procedure described in this dissertation took a long time for the practical speech coding experiment; investigation of methods to speed training would be especially useful. As a starting point some of the methods used to speed the usual backpropagation algorithm could be applied here.

Two obvious directions for future research are the application of this work to compression of still images and video. As has been done for FSVQ, blocks of pixels in an image are taken as the vectors, and the vectors are considered to be in a sequence in two directions, down and across. Therefore the state from the block above and the block to the left are used as input to the encoder and decoder of the current block. For video coding, the vectors are now a sequence in three directions, with the addition of the dimension of time. State information comes from the blocks above and to the left in the same frame, plus the same block in the previous frame (and perhaps its surrounding blocks). It is possible that the large amount of data required for such applications would increase the training time beyond practical limits. However, if training were within reason, one advantage of this method would be the allowance of a more complicated, perceptually motivated distortion measure without added encoding complexity. For video, there are no motion compensation computations; this system would depend on the internal state vectors to convey motion information between frames.

Appendix A

Perceptually Weighted Distortion Measure for Speech

This appendix gives some of the details of the perceptually weighted filter used to determine the distortion for CELP and for the speech coding experiment of Section 6.2. For further reference see Campbell et al.[1]

This distortion measure is computed one frame at a time, where a frame is 240 samples. Since the sampling rate is 8000 Hertz, this represents 30 milliseconds of sound. First $\mathbf{x}$, the 240 samples of the original speech sound, are acquired, then these samples are Hamming-windowed. From the result of that operation, an autocorrelation analysis and the Levinson-Durbin algorithm are used to obtain a 10th-order linear prediction model. Let these 10 coefficients be called

$$c_0, c_1, \ldots, c_9,$$

then a new set of 10 coefficients a_i are obtained from the c_i coefficients by a 15 Hertz bandwidth expansion, that is,

$$a_i = \gamma_A c_i, \qquad \text{for } i = 0 \text{ to } 9, \tag{A.1}$$

$$\text{where } \gamma_A = 0.994. \tag{A.2}$$

The a_i coefficients form the denominator of the linear prediction filter used by CELP,

$$\text{LP filter } = \frac{1}{A(z)}, \text{ where} \tag{A.3}$$

$$A(z) = \sum_{i=0}^{9} a_i z^{-i}. \tag{A.4}$$

Once the linear prediction filter is known, the perceptually weighted filter is found by generating a new set of 10 coefficients b_i by a further bandwidth expansion of the a_i coefficients,

$$b_i = \gamma_B a_i, \qquad \text{for } i = 0 \text{ to } 9, \tag{A.5}$$

$$\text{where } \gamma_B = 0.8. \tag{A.6}$$

Then the perceptual weighting filter is formed as

$$\text{PW filter } = \frac{A(z)}{B(z)}, \text{ where} \tag{A.7}$$

$$A(z) = \sum_{i=0}^{9} a_i z^{-i}, \text{ and } B(z) = \sum_{i=0}^{9} b_i z^{-i}. \tag{A.8}$$

The output of this filter is to be truncated at the end of the 240-sample frame, so it is convenient to express this truncated filtering operation as a 240 by 240 matrix $\mathbf{F}$. Note that a more efficient implementation will always use the actual filter $\frac{A(z)}{B(z)}$, but considering the filtering in terms of a matrix will make the formula for the distortion measure simpler, and will also elucidate an interesting result in Section A.1. The matrix $\mathbf{F}$ can be expressed by considering p_t, the impulse response of the perceptual weighting filter, which need only be found from $t = 0$ to $t = 239$. Then, since the filter is causal ($p_t = 0$ for $t < 0$),

$$\mathbf{F} = \begin{pmatrix} p_0 & 0 & \ldots & 0 & 0 \\ p_1 & p_0 & \ldots & 0 & 0 \\ \multicolumn{5}{c}{\dotfill} \\ p_{238} & p_{237} & \ldots & p_0 & 0 \\ p_{239} & p_{238} & \ldots & p_1 & p_0 \end{pmatrix}. \tag{A.9}$$

Finally, the distortion measure for a frame is the power of the filtered difference signal between the original and coded speech, which can now be expressed easily as

$$d(\mathbf{x}, \hat{\mathbf{x}}) = \|\mathbf{F}(\hat{\mathbf{x}} - \mathbf{x})\|^2. \tag{A.10}$$

The perceptual weighting filter exaggerates the effect of some frequencies on the distortion, and de-emphasizes other frequencies, based on the original signal. This is an attempt to account for so-called masking effects in hearing. However, clearly this distortion measure takes phase differences into account while it may not be so important in actual perception. While the perceptual suitability of this measure is not ideal, it is clearly better than MSE, and the complexity is certainly lower than phase-invariant distortion measures such as Itakura-Saito distortion.

A.1 Using the Perceptual Weighted Distortion Measure with Gradient Descent Training

A new filter $\frac{A(z)}{B(z)}$ must be derived for each frame. However, when training with a fixed known data set, the values of a_i and b_i may be precomputed for each frame ahead of time, then stored for retrieval during training. This precomputation could be done once (with the filters stored in a file) or could be done each time the training program is initialized (with the filters stored in memory).

When using the filter with a gradient descent algorithm such as in Section 6.2, taking the derivative of Equation A.10 reveals an interesting computational shortcut. Let the vector $\mathbf{e}$ be the result of the filtering,

$$\mathbf{e} = \mathbf{F}(\hat{\mathbf{x}} - \mathbf{x}), \tag{A.11}$$

then the error gradient at $\mathbf{e}$ is

$$\frac{\partial d(\mathbf{x}, \hat{\mathbf{x}})}{\partial \mathbf{e}} = \frac{\partial \|\mathbf{e}\|^2}{\partial \mathbf{e}} = 2\mathbf{e}. \tag{A.12}$$

Then the gradient at the decoder output $\hat{\mathbf{x}}$ is found as

$$\frac{\partial d(\mathbf{x}, \hat{\mathbf{x}})}{\partial \hat{\mathbf{x}}} = \frac{\partial \mathbf{e}}{\partial \hat{\mathbf{x}}} \frac{\partial d(\mathbf{x}, \hat{\mathbf{x}})}{\partial \mathbf{e}} = \mathbf{F}^T(2\mathbf{e}). \tag{A.13}$$

However, because of the symmetry of $\mathbf{F}$, its transpose $\mathbf{F}^T$ can be seen to represent the operation of filtering *backward* in time using the perceptually weighted filter. This suggests the following efficient algorithm for computing the gradient of distortion at $\hat{\mathbf{x}}$ for each frame:

1. Compute the vector $\mathbf{e}$ as the first 240 outputs of the perceptually weighted filter $\frac{A(z)}{B(z)}$ applied to input $(\hat{\mathbf{x}} - \mathbf{x})$.

2. If desired, compute the total distortion for this frame as $d(\mathbf{x}, \hat{\mathbf{x}}) = \|\mathbf{e}\|^2$.

3. Compute the gradient vector $\frac{\partial d(\mathbf{x},\hat{\mathbf{x}})}{\partial \hat{\mathbf{x}}}$ as the first 240 outputs of the perceptually weighted filter applied *backward in time* to input $(2\mathbf{e})$.

Bibliography

[1] Joseph P. Campbell, Jr., Thomas E. Tremain, and Vanoy C. Welch. The DOD 4.8 Kbps standard (proposed federal standard 1016). In Bishnu S. Atal, Vladimir Cuperman, and Allen Gersho, editors, *Advances in Speech Coding*, chapter 12, pages 121–133. Kluwer Academic Publishers, Boston, 1991.

[2] Joseph P. Campbell, Jr., Thomas E. Tremain, and Vanoy C. Welch. An implementation of the FS1016 4.8 Kbps CELP speech coder. Accessible on the World-Wide Web from `ftp://ftp.super.org/pub/speech/celp_3.2a.tar.Z`, 1993.

[3] Yves Chauvin and David E. Rumelhart, editors. *Backpropagation: Theory, Architectures, and Applications*. Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1995.

[4] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. John Wiley & Sons, New York, 1991.

[5] George Cybenko. Approximation by superposition of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2:303–314, 1989.

[6] Mari Ostendorf Dunham and Robert M. Gray. An algorithm for the design of labeled-transition finite-state vector quantizers. *IEEE Transactions on Communications*, COM-33(1):83–89, January 1985.

[7] Brian Flachs. *Sparse Adaptive Memory*. PhD thesis, Stanford University, Stanford, California, September 1994.

[8] John Foster, Robert M. Gray, and Mari Ostendorf Dunham. Finite-state vector quantization for waveform coding. *IEEE Transactions on Information Theory*, IT-31(3):348–359, May 1985.

[9] Robert G. Gallager. *Information Theory and Reliable Communication.* John Wiley & Sons, New York, 1968.

[10] Allen Gersho and Robert M. Gray. *Vector Quantization and Signal Compression.* Kluwer Academic Publishers, Boston, 1992.

[11] Jerry D. Gibson, editor. *The Mobile Communications Handbook.* CRC Press, Boca Raton, Florida, 1996.

[12] Robert M. Gray. *Source Coding Theory.* Kluwer Academic Publishers, Boston, 1990.

[13] Robert M. Gray and Thomas G. Stockham, Jr. Dithered quantizers. *IEEE Transactions on Information Theory*, 39(3):805–812, May 1993.

[14] Robert Hecht-Nielsen. *Neurocomputing.* Addison-Wesley, Reading, Massachusetts, 1990.

[15] John A. Hertz, Richard G. Palmer, and Anders S. Krogh. *Introduction to the Theory of Neural Computation.* Addison-Wesley, Redwood City, California, 1991.

[16] Nuggehally S. Jayant and Peter Noll. *Digital Coding of Waveforms, Principles and Applications to Speech and Video.* Prentice-Hall, Englewood Cliffs, New Jersey, 1984.

[17] Teuvo Kohonen. Improved versions of learning vector quantization. In *IJCNN International Joint Conference on Neural Networks*, pages 545–550, San Diego, CA, June 1990. IEEE and International Neural Network Society, IEEE. Volume 1.

[18] Earl Levine. Speech coding demonstration page. Accessible on the World-Wide Web from `http://www-isl.stanford.edu/people/earl/`, 1996.

[19] Earl Levine. Stochastic vector quantization, and stochastic VQ with state feedback using neural networks. In James A. Storer and Martin Cohn, editors, *DCC '96 Data Compression Conference*, pages 330–339, Snowbird, Utah, April 1996. IEEE Computer Society, IEEE Computer Society Press.

[20] Yoseph Linde, Andrès Buzo, and Robert M. Gray. An algorithm for vector quantizer design. *IEEE Transactions on Communications*, COM-28:84–95, January 1980.

[21] Stuart P. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, IT-28:127–135, 1982. Originally, unpublished Bell Laboratories Technical Note, 1957.

[22] David G. Luenberger. *Linear and Nonlinear Programming.* Addison-Wesley, Reading, Massachusetts, second edition, 1984.

[23] Donald W. Marquardt and Ronald D. Snee. Ridge regression in practice. *The American Statistician*, 29(1):3–20, February 1975.

[24] Marvin Minsky and Seymour Papert. *Perceptrons, an Introduction to Computational Geometry.* The MIT Press, Cambridge, Massachusetts, 1969.

[25] Nils J. Nilsson. *The Mathematical Foundations of Learning Machines.* Morgan Kaufman Publishers, San Mateo, California, 1990.

[26] Yoh-Han Pao. *Adaptive Pattern Recognition and Neural Networks.* Addison-Wesley, Reading, Massachusetts, 1989.

[27] Kenneth Rose, Eitan Gurewitz, and Geoffrey C. Fox. Vector quantization by deterministic annealing. *IEEE Transactions on Information Theory*, 38(4):1249–1257, July 1992.

[28] Frank Rosenblatt. *Principles of Neurodynamics; Perceptrons and the Theory of Brain Mechanisms.* Spartan Books, Washington, DC, 1962.

[29] David E. Rumelhart, James L. McClelland, and the PDP Research Group. *Parallel Distributed Processing, Explorations in the Microstructure of Cognition.* The MIT Press, Cambridge, Massachusetts, 1986.

[30] Gilbert Strang. *Linear Algebra and its Applications.* Harcourt Brace Jovanovich, San Diego, 1988.

[31] Paul J. Werbos. *Beyond Regression: New Tools for Prediction and Analysis an the Behavioral Sciences.* PhD thesis, Harvard University, Cambridge, Massachusetts, 1974.

[32] Bernard Widrow and Marcian E. Hoff, Jr. Adaptive switching circuits. Technical Report 1553-1, Stanford Electronics Laboratories, June 1960.

[33] Bernard Widrow and Michael A. Lehr. 30 years of adaptive neural networks: Perceptron, madaline, and backpropagation. *Proceedings of the IEEE*, 78(9):1415–1442, September 1990.

[34] Bernard Widrow and Samuel D. Stearns. *Adaptive Signal Processing.* Prentice-Hall, Englewood Cliffs, New Jersey, 1985.

[35] Lizhong Wu and Frank Fallside. On the design of connectionist vector quantizers. *Computer Speech and Language*, 5(3):207–229, July 1991.

[36] Lizhong Wu and Frank Fallside. Source coding and vector quantization with codebook-excited neural networks. *Computer Speech and Language*, 6(3):243–276, July 1992.