T-4385

Image Classification Applied to Seismic Semblance Data

by

Serge David Benoliel

A thesis submitted to the Faculty and the Board of Trustees of the Colorado School of Mines in partial fulfillment of the requirements for the degree of Doctor of Philosophy (Geophysics).

Golden, Colorado

Date 3 Sept. 93

Signed: S. D. Benoliel
Serge David Benoliel

Approved: Ronald N. Knoshaug
Dr. Ronald N. Knoshaug
Professor of Geophysics
Thesis Advisor

Golden, Colorado

Date September 3 1993

P R Romig
Dr. Phillip R. Romig
Professor and Head,
Department of Geophysics

ABSTRACT

Classical vision studies have demonstrated that recognition of image patterns is hierarchical. First, local features are extracted from the input image containing data elements of low-level information. Then, those features are combined into higher-order features with data elements of higher-level information. This vision paradigm can be built into a layered, hierarchical, and highly-constrained backpropagation neural network for the purpose of classifying images. I apply this technique of image classification to semblance data as the first step in an automated process for picking seismic stacking velocities. In the selection of velocities from semblance images, additional information is needed as a guide to determine the correct velocity picks. One example of such information is the classification of semblance images into classes that may correspond to different geological sections. Stacking velocities are then determined using the classification information as a constraint for a computer-search algorithm. I illustrate the use of this computerized, automated velocity picking technique with computer generated, laboratory model, and field data, demonstrating that velocities picked automatically yield results equivalent to those picked manually. Because of the time required to train the network, and the high density of semblance information needed, the use of automated velocity picking may not be economically viable for routine data processing. When, however, densely sampled information is required, such as in AVO analysis, the automated scheme described here is a cost effective alternative to picking velocities by hand.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

INTRODUCTION

The goal of seismic data processing is to produce an image of the earth's subsurface from seismic experiments recorded in the field. Seismic experiments create disturbances (e.g. explosions) at the surface of the earth that lead to the propagation of **seismic waves**[1] inside the earth. These seismic waves are recorded over a fixed amount of time by **geophones** located at some fixed distances from the source of the disturbances. The time history of displacement observed at each geophone is referred to as a **seismogram**.

When processed, these seismograms yield a pseudo-image of the earth. I use the term "pseudo-image" because the vertical axis of the image does not represent the depth but the recorded time. The most fundamental step in seismic data processing is the **stacking process** that uses the assumption that desired seismic events or **reflections** moveout hyperbolicly when seismograms are sorted by source-receiver offset. To obtain the image of the earth's subsurface, the velocity field through which the seismic waves propagate, must be known or estimated directly from the data. This leads to the following paradox: seismic data processing yields the correct earth model only if the earth model is known a priori. Usually, images of the earth's subsurface are constructed using many velocity models. The "best" of these images is usually chosen from a criterion that is expressed in terms of reflector coherence. That is, the correct velocity field is the one that produces the most visually appealing image. This criterion can be quantified by first generating **semblance images** that measure

[1]A glossary of the terms written in bold is given in Appendix C or D.

the **coherence** of seismic data that moves out with distance hyperbolicly at different **stacking velocities**, and then selecting the coherence at amplitude maxima.

Unfortunately, for a variety of reasons the "best" stacking velocity is not always associated with maximum coherence amplitudes. For example, reflected arrivals do not always moveout hyperbolicly: hyperbolic moveout strictly holds for a flat or dipping single reflector earth model. Also, there are a few unwanted seismic events having high coherence amplitudes, that mask the coherence amplitudes corresponding to the wanted reflections. Examples of these are **multiples** and **diffractions**. To illustrate these complications, consider the following experiment: two stacked sections are generated from a laboratory model (to be described later), using velocities selected by hand (Figure 1.1-a) and using the velocities associated with the maximum coherence amplitude (Figure 1.1-b). The stacked section processed with velocities picked by hand shows sharper reflections with better continuity. Thus, the velocity field chosen by hand produces a visually more appealing section than the one that uses maximum coherence amplitudes.

If the correct velocities cannot always be derived from the maxima, then what techniques are available for selecting the correct velocities? In the example described above, I used a priori knowledge of the geologic section reproduced in the model to choose my stacking velocities. Use of a priori geologic information is not atypical for this problem, rather it is the norm. Because the way of picking velocities depends on the geology of the subsurface, then the previous question is rephrased: can an automated velocity picking scheme be designed that incorporates knowledge of the subsurface geology?

Selection of the correct coherent events in semblance images can be considered as a **pattern recognition** problem. There are no "cut-and-dry" rules for selecting the coherent event that will yield the correct velocities. Instead, we must rely on intuition based on past experience to select those velocities.

Because the the manual technique for velocity picking can involve large amounts of data, velocity picking can become quite laborious and time-consuming especially in the case of continuous velocity analysis for A.V.O. studies. Thus, the automation of velocity analysis could greatly enhance the efficiency in processing seismic data for such studies. With the added feature that coherence amplitude selection is a **pattern recognition** problem, I employ the popular technique of **neural networks**.

Neural networks are I/O systems made up of many **nodes** that are connected to each other. The nodes are organized in **layers**: the **input layer, the output layer** and the layers between the input and the output layer that are designated as **hidden layers** [9]. The **connnections** between nodes in different layers define the structure of the neural network and are described by numerical scalar values called **weights**. Neural networks are **learning** systems by which the weights are determined during a **training phase** using **training samples** [9].

In recent years, neural networks have been applied to seismic data processing on an experimental basis for the solution of difficult problems such as seismic **trace editing** ([12], [17]), and **first-break picking** ([17], [26] and [5]), and have even advanced to the production stage. J. Schmidt [22] tried to solve the problem of **velocity picking** using semblance images by first picking maxima using a simple network in the horizontal, then the vertical direction of the images. He then selected the maxima that were closest to a reference velocity function. To confirm the validity of his picks, he measured the distance between the reference velocity and his newly selected velocity function using another neural network. Input to this network consisted of semblance data derived from the new velocity function. Although he was successful in picking velocities from field data, the last step in checking the validity of the picks is questionable because he used too few training samples to train a large network.

The networks described above were of simple design and required minimal training. Namely, they consisted of networks made up of a few nodes and using training sets with a few input **patterns**. The simplicity in the design of these networks stems from the fact that they were developed to look at 1-dimensional patterns. The problem under consideration, however, is a 2-dimensional one. In this research, I follow the work of Lecun [13] and I develop a 2-dimensional neural network that takes semblance images directly as input and classifies them according to their geometrical **feature** patterns.

Initially, the process of velocity picking can be posed as a **mapping** problem where the input to the network is a semblance image and the corresponding output of the network is an image of the same size as the input but with non-zero pixels only at the exact locations of the velocity picks. In other words, the output image shows the locations and the number of velocity picks for each semblance image. Because the images are constructed from many data pixels this mapping from the input image to output image creates an excessively large network with too many weights. Furthermore, large network size tends to impair the **generalization** process causing the testing of a trained neural network with unseen data to fail [1].

To avoid creating an excessively large network, I reformulate the velocity picking problem. Instead of considering it as a mapping problem (from image to image), I change the formulation to a **classification** problem (from image to class). In the classification process, the number of elements in the output is reduced with respect to the number of elements in the input image. As a result, the size of the network is correspondingly reduced. Even though the network size is reduced by modifying the output, it is still too large for properly training the neural network using only a few training samples. I show in Chapter 3 that the size of the network can be decreased further by setting up constraints and building a priori knowledge into the network. It is shown that such a network structure achieves good **generalization performance**.

The classification network by itself does not perform velocity picking. Instead, I use the output classification from the network as a guide for a computer search algorithm to locate the correct velocity picks (see Chapter 4).

The remainder of this thesis is organized as follows:

Chapter 2: - Background about velocity picking and neural networks are given.

Chapter 3: - A simple example is provided to show how to insert constraints into a classification network.

Chapter 4: - A procedure is presented on how to pick velocities using classified output images.

Chapter 5: - The velocity-picking procedure as described in Chapter 5 is applied to modeled data.

Chapter 6: - The same procedure is applied to synthetic shot records.

Chapter 7: - The same procedure is applied to field data.

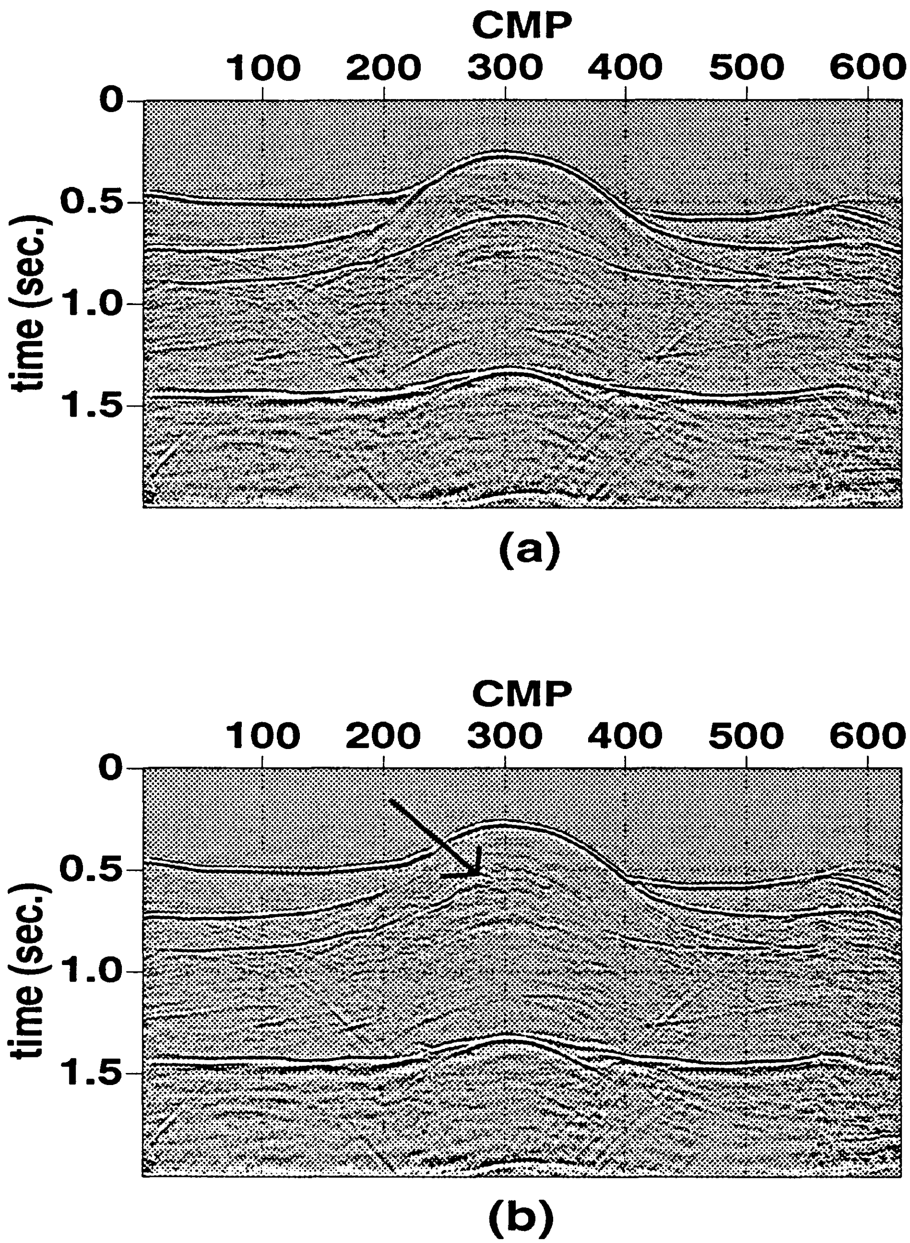


Figure 1.1. Comparison between two stacks: the first stack (a) is processed with stacking velocities picked by hand employing some geologic insight; the second stack (b) is processed with velocities emanating from the maximum semblance peaks. The arrow in (b) shows the location where the difference in reflection continuity is the most obvious between the two stacks.

Chapter 2

BACKGROUND

As described in the introduction, the quality of the image produced from seismic observations is critically dependent on the stacking velocity field used in its construction. Although we have mentioned the use of semblance images to aid in the estimation of stacking velocity, other estimation techniques are also routinely used. Each of these techniques fundamentally looks at the same quantity, the stacked reflector coherence, but the information is presented to the data processor in different domains. The three domains considered are: constant velocity stacks, unstacked moved-out **common midpoint gathers**, and semblance images. Below, I briefly describe each domain and list their advantages and disadvantages.

2.1 Velocity Analysis Domains

Conceptually, the simplest domain to work in is the constant velocity stack domain. Here, the data are corrected for normal move-out, **stacked** at many constant velocities, and then compared visually for **reflection** continuity [30]. The final selected velocity field is then chosen as a function of time and position and applied to produce the stacked section. By working in this domain, the user has the advantage of seeing the reflections as they would appear in the final **stacked section**. One drawback is that this technique generates a very large amount of data that can be difficult to manipulate, particularly in an non-automated scheme. Another disadvantage is that the "quality" of the velocity picks, which is based on the visual appeal of the stacked sections, cannot be quantified.

Unstacked moved-out CMP gathers are used to check visually how well the moveout process (NMO) has performed, namely, whether or not the hyperbolicly moved-out reflections have been corrected enough for the stacking process [30]. Visual checking is, however, subjective and qualitative. As in the constant velocity domain analysis, corrected CMP gathers are typically generated for a series of constant stacking velocities. Use of velocities that are too low overcorrects the observed moveout. Velocities that are too large undercorrect the observed moveout. The final stacking velocity field is chosen as a funtion of time and position from these constant velocity gathers. Because the use of CMP gathers is so closely related to the constant velocity domain, the disadvantage for the use of that domain also holds here.

The final domain, which has already been described in the introduction, is the semblance domain. Semblance is a multichannel coherence measurement for stacked seismic reflections along hyperbolas of different seismic velocities (see [24] and Appendix A for details of the computation of semblance). One advantage of using semblance images is that the amount of data generated is not as large as that generated using the constant velocity stack or the CMP gather domains. This domain, therefore, allows the user to test *more* velocities at *finer* increments than in the constant-velocity stack or CMP gather domains. Another important advantage of this domain is that it gives a quantitative measurement of reflection coherences along hyperbolas since coherence amplitudes usually peak at a particular velocity. The drawback in using this domain is that the user is *not* able to see the reflections as they appear in the final stack section.

The most compact and quantitative information pertaining to stacking velocities is, therefore, provided by semblance analysis. Typically, semblance values are plotted as a function of stacking velocity and record time for analysis. Thus, scalar information presented in this manner can be considered as an image of the appropriateness of a particular stacking velocity at a given record time. To quantitatively analyze this image as a part of an automated velocity picking scheme, I develop neural networks capable of classifying semblance image into known geological sections.

2.2 Neural Network

Neural networks are I/O systems made up of many **processing** or **computing elements (nodes)** that are either **fully, locally** or **sparsely connected** to each other. The nodes are organized in **layers**: the **input layer**[1], **the output layer** and the layers between the input and the output layer designated as **hidden layers** [9].

The **connnections** between the nodes in different layers define the structure of the neural network and are described by numerical scalar values called **weights**. **Biases** are types of weights describing the connections between "dummy" nodes (having a constant value of one) and all computing elements [9] (see Figure 2.1). Thus, a quantitative description of the network architecture is made by using a set of properly indexed[2] weights and biases.

Two operations are performed in sequence on each computing element of a neural network (see Figure 2.2). The first operation is the dot-product of the input vector $\vec{o}$ with its associated **weight vector** $\vec{w}$. In the second operation, the resulting dot-product is scaled (amplitude is changed) using a **squashing function**[3], yielding a nonlinear relationship between the node output and the input vector. Because of the

[1]The input layer is not counted in the description of the neural network since it is made up of nodes that are *not* computing elements.

[2]There are two subscript indices for each weight and bias. The first index denotes the destination node number and the second index denotes the source node number, i.e. weight w_{ji} which describes the connection from node i to node j. For a bias the second index is always equal to zero i.e. w_{j0}.

[3]The most common squashing function is the **sigmoid function** (see Appendix B) expressed as $f(x) = \frac{1}{1+e^{-x}}$.

cascade of node computations, the relationship between input and output vectors in a multilayer neural network, is, therefore, highly nonlinear [9].

Neural networks can be categorized by the way they transform the input data, into one of three types [9]: **pattern classifier** networks place input patterns into one of several predefined classes; **pattern transformer** networks map input patterns into an output pattern of the same size; **time sequence predictor** networks extrapolate an input time series to some future time. Because this study is concerned with the classification of images according to their geometrical feature patterns, I use only a pattern classifier network to process information.

Neural networks are capable of learning. Learning consists of adjusting the network weight and bias values until a learning criterion has been reached. There are three types of learning [9]: **supervised learning** that uses as its learning criterion the minimization of the difference between the actual and expected outputs; **unsupervised learning** that does not rely on expected outputs; **reinforcement learning** in which the only feedback indicates whether or not each output is correct.

Because in this research, feature patterns defining each class are known in advance, I employ the supervised learning paradigm to adjust the network weights.

A **supervised learnable system** is one that tries to change itself in order to closely match an ideal system in which a given input returns an "expected" or desired output [9]. Let the inputs, outputs, and desired outputs of a system, be defined as $\vec{I}_1, \vec{I}_2, \vec{I}_3, .., \vec{I}_p, .., \vec{I}_P$, $\vec{O}_1, \vec{O}_2, \vec{O}_3, .., \vec{O}_p, .., \vec{O}_P$, and $\vec{D}_1, \vec{D}_2, \vec{D}_3, .., \vec{D}_p, .., \vec{D}_P$ respectively, and the weights connecting the nodes in the network be given by a weight vector $\vec{w}$. Learning consists of finding the vector $\vec{w}$ that minimizes the average **mean-square difference** (or **l-2 norm**), $C(\vec{w})$, between the actual outputs and the desired outputs over a set of P **training samples** as described by the equation below:

$$C(\vec{w}) = \frac{1}{P}\sum_{p=1}^{P} ||\vec{D}_p - O_p(\vec{w}, \vec{I}_p)||. \tag{2.1}$$

While it may be possible to obtain low $C(\vec{w})$ values for a particular training set, this does not guarantee good fits if members of the training set are changed. **Generalization** is the ability of a trained system to produce correct outputs using inputs that are *not* in the **training set**. Generalization in the context of pattern recognition is analogous to interpolation and extrapolation in curve fitting. Good generalization is an indicator that the neural network has learned well. On the other hand, **memorization** is referred to as the inability of a trained system to give correct outputs when input **patterns** applied to the system are *not* in the training set.

The process by which the neural network learns is performed in two phases:

1. **The training phase:** The neural network learns with a training set of input/expected output pairs. Training is performed by adjusting weights and biases until the l-2 norm between the actual and the desired outputs (equation 2.1) is below a user-defined threshold. Another criterion for stopping the training phase is to compute the maximum **node-error** between the actual and desired outputs. When the maximum node-error is below a user-defined threshold, the network has properly learned. **Learning performance** is measured by the number of **epochs**[4] required for the system to learn.

2. **The testing phase:** Network **generalization performance** is tested using input samples not used in the training phase. Generalization performance is

[4]An **epoch** is a "training session" in which all the input patterns in the training set are presented to the neural network for training.

measured by comparing expected to actual outputs using either a **l-2 distance** or a maximum **node-error**. If the l-2 distance or the maximum node-error is below a user-defined bound, it is an indication that the network **generalizes** well.

The algorithm that changes the weights using the training set samples is called the **learning algorithm**. One such algorithm is the **backpropagation** algorithm [20], which is a **gradient-descent** technique in the multi-dimensional weight space (see Appendix B). In this thesis, I have chosen to use this algorithm because of its many successful applications to a variety of problems in the past[5].

[5]This learning algorithm may terminate under **local minima** conditions without ever reaching the optimum learning solution (the **global minimum**).

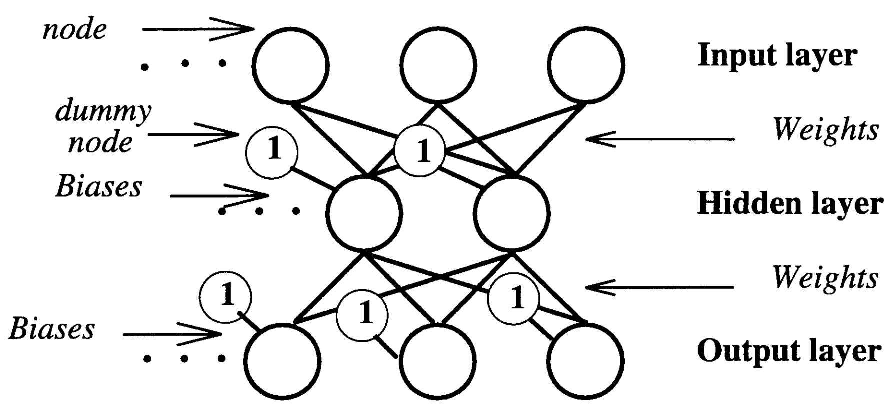


Figure 2.1. Basic architecture of a two-layer neural-network system consisting of one input layer, one hidden layer and one output layer. Layers are made up of nodes. Connections between nodes in different layers are described by weight values. Biases are weights with input values always equal to one.

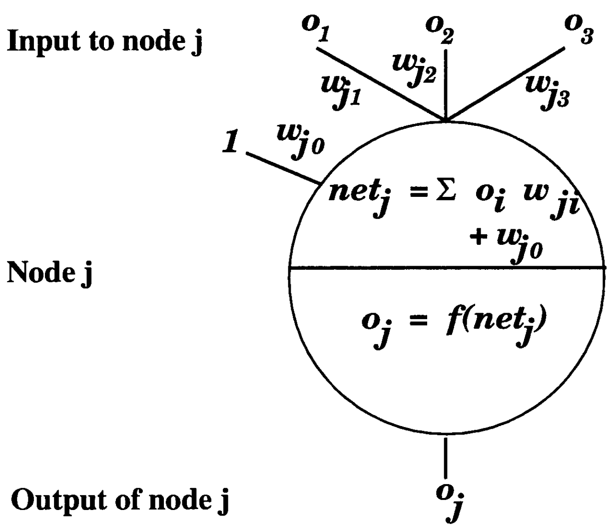


Figure 2.2. Two operations are applied to a neural-network node j which receives inputs from three nodes and generates the output o_j: First, computation of net_j or the dot-product between input vector $\vec{o} = (1, o_1, o_2, o_3)$ and weight vector $\vec{w} = (w_{j0}, w_{j1}, w_{j2}, w_{j3})$; Second, scaling of net_j through the squashing function f. The component w_{j0} of the weight vector $\vec{w}$ is also called the bias.

Chapter 3

IMAGE CLASSIFICATION

Using a simple example, I show how to design a complex, high-performance classification network that takes images directly as inputs. Starting with a one-layer network, more complexity and constraints are introduced. The constraints are necessary to decrease the number of independent weights or **free parameters**, leading to improved generalization performance.

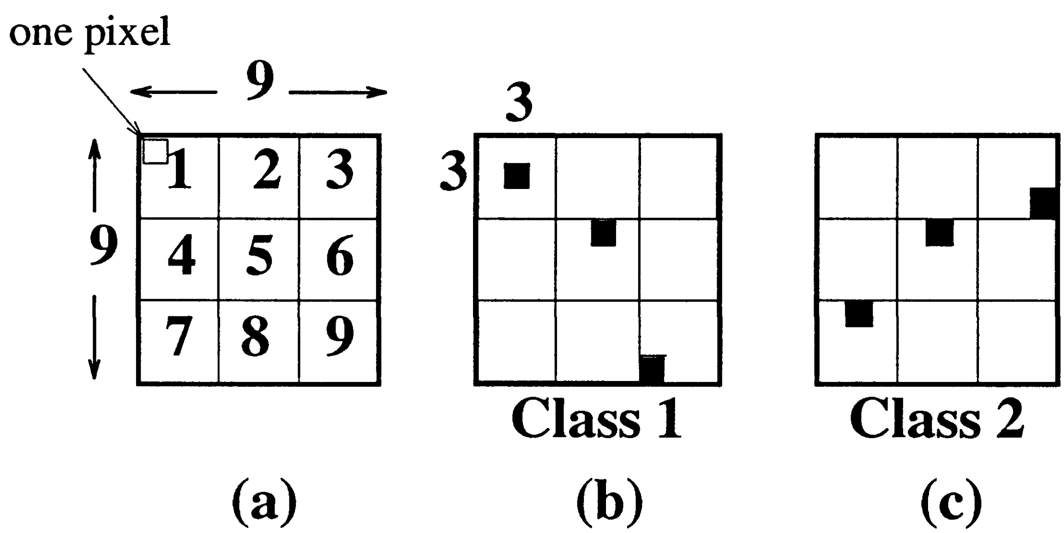


Figure 3.1. Partition of the input image of (9 x 9) pixels into nine nonoverlapping areas of (3 x 3) pixels (a). Examples of data belonging to two different classes (b) and (c).

For the testing of the neural network architecture, an image data set is constructed that has 2 classes. The images consists of a (9 x 9) binary pixel pattern divided into 9 nonoverlapping areas (see Figure 3.1-a). The numbering scheme for these 9 areas, used in the following discussion, goes from 1 in the upper left corner of the image, to 9 in lower right corner of the image, in a left-to-right and a downward manner (Figure 3.1-a). Class 1 is defined as having one black pixel (dot of unit amplitude) in each of the squares of the main, dextral diagonal (see Figure 3.1-b). Class 2 is defined the same way, but with black pixels along the main sinistral diagonal (see Figure 3.1-c). Based on all possible locations of the three black pixels, there are 729 or (9 x 9 x 9) possible images in each class, or a total of 1,458 possible images from either class 1 or class 2.

The goal is to now design a neural network that can classify these images into class 1 or class 2. To do this, four types of neural network architecture are tried: 1) a one-layer, **fully-connected** neural network where the nodes in the output layer are connected to all the nodes in the input layer, 2) a two-layer, fully-connected neural network where the nodes in one layer are connected to all the nodes in the preceding layer, 3) a three-layer, **partially-connected** neural network where the nodes in a destination layer are connected to only a neighboring subset of nodes in the source layer, 4) a three-layer, partially-connected and constrained neural network where in addition to the characteristics of the previous type, some equality constraints are imposed to the connections. The architecture of each of these types of network is shown schematically in Figure 3.2.

For each neural-network architecture, images are directly entered into the network as input, and the corresponding output from the network indicates the class to which each image belongs. With a (9 x 9)-pixel input image, the number of input nodes is 81, and there are 2 nodes in the output layer defining the output class pattern. If the input image belongs to class 1, the desired output-node values are [1, 0]. If the input image belongs to class 2, the desired output-node values are [0,

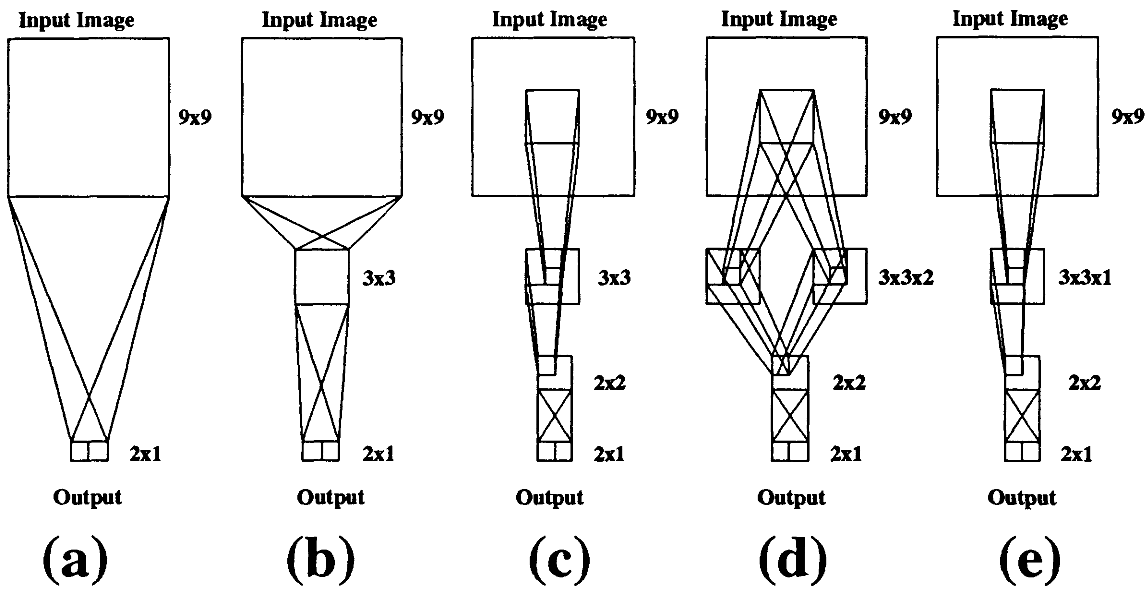


Figure 3.2. Four type of network architecture: (a) one-layer fully connected; (b) two-layer fully connected; (c) three-layer partially connected; (d) and (e) three-layer constrained and partially connected.

1]. Before the images are entered as inputs into the network, their amplitudes are scaled to fall within the range of [-0.5,0.5]. This type of symmetric scaling speeds up the network **convergence** during the training phase [27]. Fifteen samples from each class are used during the training phase. The remaining images in the data set (1,428 images) are used to test network generalization.

To test the network's ability to learn from the training set, a learning criterion is constructed using a measure of the misfit between the actual class output to the expected class output for every training image. One of several different types of misfit measurements can be used. For example, we could use the l2-norm, the euclidian l-2 distance or the maximum absolute difference. The test on the maximum absolute difference is more severe than that on the l2-norm or the l-2 distance. That is, if normalized between 0 and 1, the maximum absolute difference is always the estimate of the misfit. I will, therefore, use the maximum absolute difference as my learning criterion. The network is said to have learned when the maximum of all the absolute differences between the actual output-node values and the desired output-node values over the entire training set, is below a user-defined threshold. With 30 images in the training set, I, therefore, compute 60 absolute differences or **node errors**. Because the two types of patterns I want to classify, are very different, I choose a rather high learning node error threshold of 0.2[1].

Once the network has learned from the training set, the remaining images in each class are used to evaluate the generalization performance. In measuring this performance, I compute the **l2-distance** (or **root-mean-square difference**)[2] between the actual and the desired output of the network. When the l-2 distance is less than a user-defined threshold, the network is considered to have produced the correct classification for that input image, namely, a **good detection**. When the l-2 distance is greater than the threshold value, the image pattern is not classified correctly, namely, a **false alarm**. For the measurement of generalization performance, I use a more relaxed criterion than that for training performance. For this case, I choose a generalization l-2 distance threshold equal to 0.2. Since I use a generalization l-2 distance criterion of 0.2, a generalization performance below 90% (i.e. 90% of good detections in the testing set) is not considered. That is, for a network to generalize in an acceptable fashion, it must be able to correctly classify at least 90% of the input images during the generalization tests.

[1]The range of values I can choose for the learning node error threshold varies between 0 and 0.5. A value of 0.0 means that the image fits perfectly to its class. A value of 0.5 or above does not have any classification meaning. The value of 0.2 seems to be a good compromise.

[2]The **root-mean-square difference** is defined as $\sqrt{(o_1 - d_1)^2 + (o_2 - d_2)^2}$ where o_1, o_2 and d_1, d_2 are the actual and desired outputs for each output node.

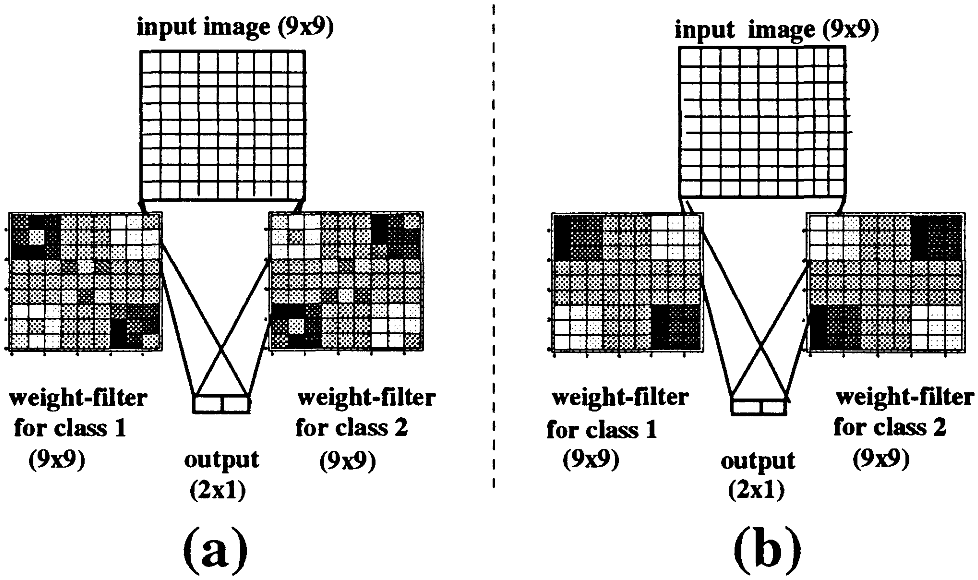


Figure 3.3. Weights for the one-layer, fully-connected neural network after learning with a training set of 30 images (a), and after learning with the whole data set (1458 images) (b). (White shading represents the highest positive values and black shading represents the lowest negative values).

3.1 Design 1: One-layer, Fully-Connected Neural Network

The simplest system is a one-layer, fully-connected neural network. The eighty-one pixel image is applied as input to the neural network (see Figures 3.2-a and 3.3). The output layer has two nodes (corresponding to two classes) that are fully, and directly connected to the input layer. This design provides a weight structure of 162 weights plus 2 biases (or 164 free parameters).

Using a training set of 30 images and a maximum node error of 0.2, this network learned successfully in 120 iterations, or 4 epochs[3]. Good learning, however, does not guarantee good generalization. In measuring the generalization performance of

[3]If a one-layer neural network is able to learn, that means the relation between input and output is likely to be linear [3]. This relationship does not occur in trained multilayered neural networks.

the neural network with the remaining of the image set, I find that only 72% of the images are classified correctly according to a l-2 distance threshold of 0.2., which is far below the 90% minimum, fixed as a goal.

This poor generalization performance can be understood by examining the weight patterns of the trained network (Figures 3.3-a and 3.3-b). In Figure 3.3-a, I show the weight patterns in a (9 x 9) grid when the 30 image training set is used for learning. In Figure 3.3-b, I show the weight patterns obtained by the network when the full, 1458 image data set is used to train the neural network. By examining the weight matrices of the output units in Figure 3.3-a, the first output node is seen to receive negative weights from regions (1) and (9) of the input image and positive weights from regions (3) and (7) of the input image (see Figure 3.1-a for the numbering system that describes the regions in the input image). The second output node receives positive weights from regions (1) and (9) and negative weights from regions (3) and (7). Regions 2, 4, 5, 6, 8 have weights that are close to zero as indicated by grey shading in Figure 3.3. Areas 2, 4, 5, 6 and 8, therefore, have little effect in the classification process. The remaining areas 1, 3, 7 and 9, on the other hand, exert significant influence. Thus, I conclude the learning process generated weight patterns that can be described as a set of **weight filters** tuned to recognize a pattern "averaged-out" from the combination of all training samples. In other words, the network "over-learned" the training examples. The network has memorized the training patterns and its generalization performance, in effect, is disappointing.

3.2 Design 2: Two-layer, Fully-Connected Neural Network

To try to increase the generalization capability of the network, a hidden layer of dimension (3 x 3) nodes is inserted between the input and the output layers as shown

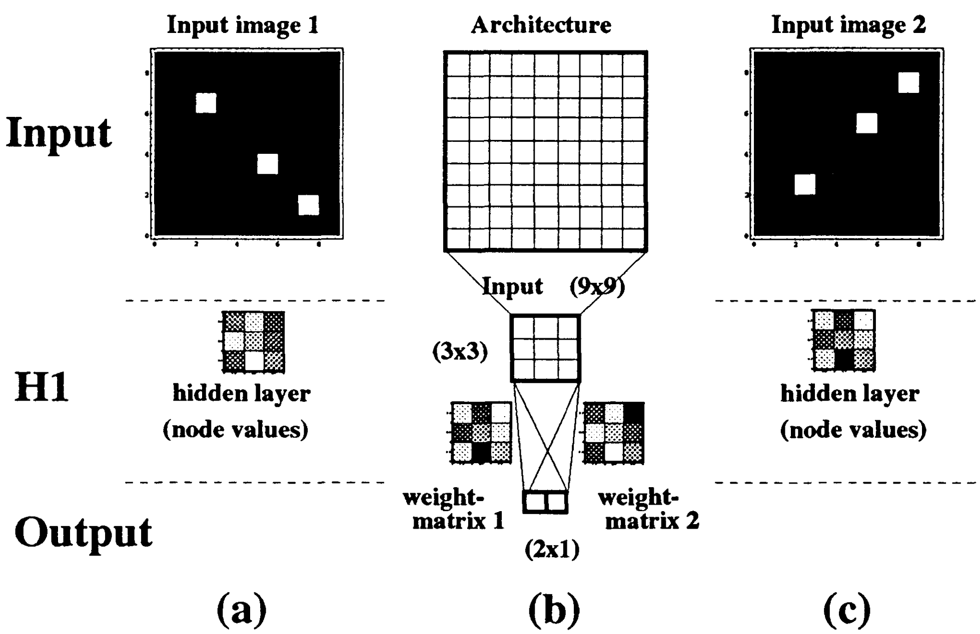


Figure 3.4. Schematic model of the two-layer architecture with its two internal weight matrices producing a two-node classified output (b). Examples of (3 x 3)-node values for the hidden layer corresponding to two different (9 x 9)-pixel input images (a) and (c).

in Figure 3.4 [27]. The network is fully connected from the input layer of 81 nodes to the hidden layer of 9 nodes and from the hidden layer to the 2-node output layer. Thus, the number of weights for this two-layer system is 758, a *marked* increase from 164 for the one-layer system.

Applying the previously described learning and testing algorithm to the two-layer design, the network requires 180 iterations or 6 epochs to learn, and the generalization performance measured for this system is 89%. Because the network has such a large number of weights while it is being trained with only a few images, the generalization performance is still not reliable [2].

With a 30-sample subset used to train the highly overparameterized 758-weight, two-layer neural network, one would expect a degradation in its generalization performance [2] with respect to the performance of the one-layer neural network. On the contrary, I find an increase in performance, from 72% to 89%. To understand this apparent anomaly, consider a plot of the hidden-layer node values for different input class images (Figures 3.4-a 3.4-c). The first layer of the network acts as a data reduction system, compressing the input images from a (9 x 9) grid to a (3 x 3) grid. The actual classification occurs between the hidden layer and the two-node output layer. The number of free parameters between the hidden layer and the output layer is equal to 20 (18 weights plus two biases). Thus, in effect, there is an actual reduction in the number of free parameters, from 164 to 20 in the actual classification network, thereby accounting for the improvement in the generalization process.

3.3 Vision Paradigm

The previous network was over-parameterized (758 weights for only 30 training examples), and as a result it still generalized poorly. Therefore, we must design a network architecture with fewer free parameters to yield a greater generalization performance. One way to decrease the number of free parameters is to incorporate a priori knowledge about the classification process into the architecture of the network. An example of such a priori knowledge is illustrated by classical vision studies that have demonstrated that recognition of patterns is **hierarchical** [11], [14]. In the first hierarchical level of the vision model, **local features**[4] that are of distinctive geometrical shape and of limited area in the input image, are extracted from the input image by means of **feature detectors** for the purpose of creating **feature maps**. In the context of an artificial vision system, a **feature detector** is a two-dimensional (or spatial) operator applied to subsets of the input image for the purpose of detecting the presence or the absence of a particular feature. The resulting feature map indicates the approximate *locations* of a specific feature within the image. Then in the next hierarchical level, features extracted from the feature maps are combined to form higher-order feature maps (i.e. maps with higher density of information) and so on...

Based on this vision model and following the work of LeCun [13], the next networks tested use hierarchical-level design. The vision paradigm can be built into the architecture of the network by mapping local features from the input image into one or more feature maps in the first hidden layer[5]. Then, feature maps in the first hidden layer are combined to form higher-order feature maps in the second hidden layer. This paradigm is extended further by inserting more layers in the same hierarchical manner as described above thus creating higher and higher-order features maps.

These types of networks also differ fundamentally from those tested previously in another respect. In the previous network designs, the nodes in a destination layer

[4]The same local feature may appear many times in different locations inside the image.

[5]For every type of feature, a different feature detector must be used.

are connected to all of the nodes of a source layer (i.e. **fully-connected** network). In the new design, in order to isolate a particular feature, and decrease the number of free parameters, all nodes within different subsets of nodes in the source layer are connected to one node in the destination layer. This type of connected network is referred to as **partially-connected**.

In this vision-model type network, local features are detected using a fixed-dimensioned **scanning** window that is moved over the input image in fixed increments, simultaneously building feature maps in the first hidden layer H1, by vector-multiplying each **receptive field** (in the input layer) with its associated weight filter [13]. The scanning process allows each node in a destination layer to be connected only to a subset of the nodes in a source layer. A receptive field is a set of nodes such that the output of each node is a measure of the image intensity over a limited area of the vision field. In an artificial vision system (neural network), it is a subset of neighboring nodes within the source layer. As a result of the above scanning process, a pyramid-structured network with partial connectivity between layers is constructed (see Figures 3.2-c and 3.5-b).

In the following network designs, I consider two types of partial connectivity between layers: 1) a **nonweight-sharing** partial connectivity that consists of changing the set of weights associated with each receptive field as the source layer is scanned, 2) a **weight-sharing** partial connectivity that consists of maintaining the same set of weights associated with each receptive field as the source layer is scanned.

3.3.1 Design 3: Three-Layer Neural Network with Nonweight-Sharing Partial Connectivity

This design consists of two hidden layers H1 and H2 from which feature maps are constructed based on a nonweight-sharing partial connectivity structure (see Figure 3.5-b). Thus, for a (9 x 9)-pixel image applied as input to the network (see Figure

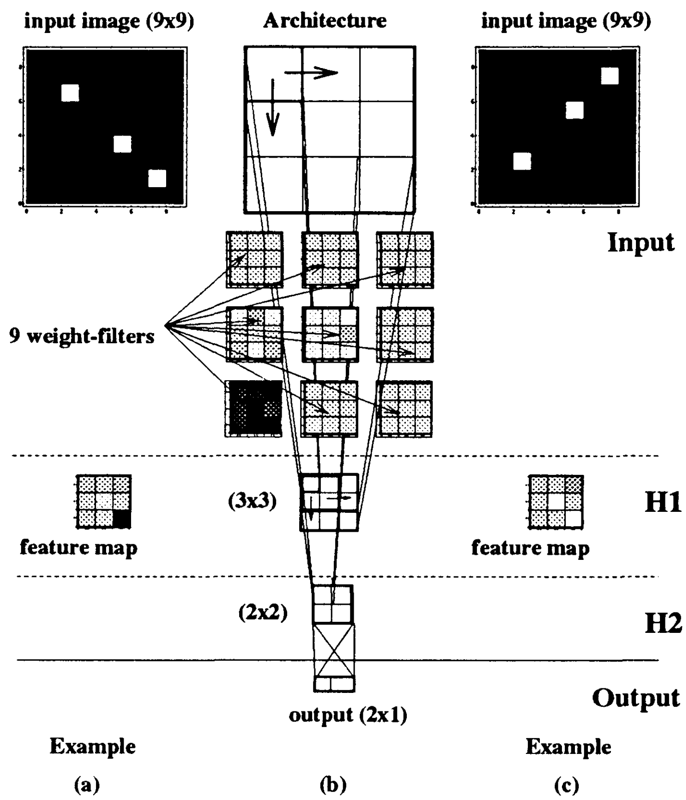


Figure 3.5. A locally-connected network architecture and associated weight filters (b). Examples of feature maps in H1 for two different classes of images (a) and (c).

3.5-b), I design nine independent weight filters, each associated with a (3 x 3)-pixel input image subset, or receptive field. In this particular design, the receptive fields in the input image do not overlap. By vector-multiplying a receptive field with its associated weight filter, one node in the H1 feature map is generated. As the entire input image is scanned, we construct a (3 x 3)-node feature map in the H1 layer. The second-layer feature map, H2, is built by scanning the (3 x 3)-node feature map in H1 using a window of (2 x 2)-node size. The size of the receptive fields in H1 is, thus, (2 x 2) nodes. In the scanning process, two consecutive receptive fields in H1 overlap either by one node in the X-direction or one node in the Y-direction. As the entire H1 feature map is scanned, I create a (2 x 2)-node, higher-order feature map in the H2 layer. In the final part of my design, H2, the second hidden layer, is then fully connected to the two-node output layer.

The resulting network with 120 connections is found to learn in 1,560 iterations or 52 epochs. Therefore, although the number of weights is decreased significantly by this design, the time requirements for training the network has significantly increased. The generalization performance, however, now reaches an acceptable level of 94%, far better than either of the two networks previously tested. By examining the weight filters (see Figure 3.5-b), it is interesting to note that only one filter out of nine is active in the classification process. The receptive field of the active filter is at the lower left corner of the input image. The detection or the nondetection of the dot intensity in this corner of the input image is sufficient to classify that image into one of the two image classes.

3.3.2 Design 4: Three-Layer Neural Network with Weight-Sharing Partial Connectivity

Because the same local feature can appear at different positions in the input image, a feature detector should detect such a feature anywhere in that image (designated as shift-invariant detection). The detection of the same feature in different locations in the input image is implemented by constraining all receptive fields in the input layer to have equal weights. This technique is called weight-sharing because all the receptive fields *share* the same set of weights. As a result, the number of free parameters in the network is reduced since *each* weight controls *many* connections at different locations inside the input image.

As in the previous design, the architecture of this design consists of two hidden layers H1 and H2 (see Figure 3.6-b). H1 and H2 are constructed using the weight-sharing partial connectivity technique. Layer H1 contains two feature maps and layer H2 is made up of only one feature map. I build each of the two feature maps in H1, by convolving the input image by a constant weight filter of (3 x 3)-node size. The receptive fields in the input layer do not overlap and each feature map in H1 has a (3 x 3)-node size. A hidden-node value in the second hidden layer H2, is obtained by first juxtaposing two receptive fields, one from each feature map in H1, and then convolving the combined fields with its associated weight filter. In the scanning process, the receptive fields in each H1 feature map overlap either by one node in the X-direction or one node in the Y-direction (see Figure 3.6-b). Layer H2 is, thus, composed of one (2 x 2)-node, high-order feature map. As in design 3, H2 is then fully connected to the output layer.

The resulting network has 210 connections but only 58 free parameters. With such a structure, the network learns in 1,020 iterations or 34 epochs, and the generalization performance is found to reach the very acceptable level of 97%. By examining the two feature maps of the first hidden layer notice, however, that they look similar (see Figure 3.6-a and Figure 3.6-c). Therefore, the two weight filters between the input layer and H1 appear to be performing the same task. This suggests that the network would perform just as well with one feature map as with two feature maps in the first hidden layer. Consequently, one filter should be sufficient to design the hidden layer, H1, thereby reducing the number of free parameters to 36. After constructing and testing such a network, I find that the network learns in 1200 iterations or 40 epochs, and the generalization performance reaches 100% .

In summary, I examined four types of network architecture (see Figure 3.2) and demonstrated procedures for analyzing these architectures in order to develop improved network designs yielding better generalization performance. This approach was successful, leading to a partially-connected, hierarchical multilayer architecture (based on biological vision models) that produced 100% generalization performance. The performance results of all four types of architecture (summarized in Table 3.1) shows that generalization generally increases as the number of free parameters goes down and as the architecture becomes more constrained.

Table 3.1. Generalization performance for five network architectures of increased complexity and increased constraints.

architecture	*connections*	*free parameters*	*generalization performance*	*iterations for learning*
1. One-layer fully connected	164	164	72%	120
2. Two-layer fully connected	758	758	89%	180
3. Multi-layer without weight-sharing	120	120	94%	1560
4. Multi-layer with weight-sharing (two feature maps in H1)	210	58	97%	1020
5. Multi-layer with weight-sharing (one feature map in H1)	120	36	100%	1200

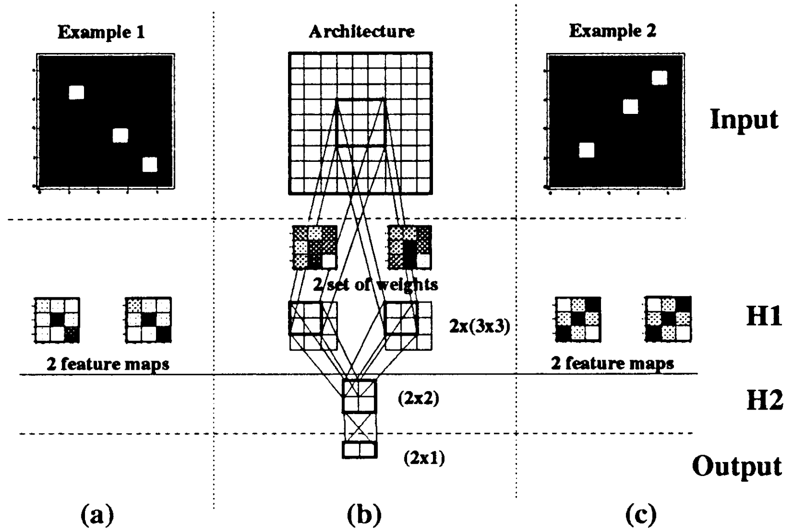


Figure 3.6. A shared-weight network architecture with two feature maps in H1 and two sets of weights from the input layer to the first hidden layer (b), and examples of feature maps in H1 for two classes of images (a) and (b).

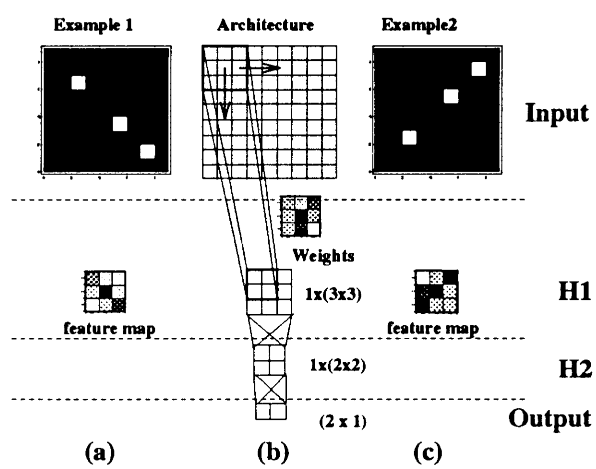


Figure 3.7. A shared-weight network architecture with one feature map (b) in H1 and examples of feature maps in H1 (a) and (c).

Chapter 4

AUTOMATED VELOCITY PICKING

Having explored the ability of a neural network to classify simple images reminiscent of semblance images, I will now proceed to design a processing package capable of automated velocity picking. In the previous chapter, neural networks were used only to classify images into specific types. Although useful, the classification of a semblance image into a specific type is not the desired result. Rather, the establishment of a stacking velocity function from the semblance image is the desired result. Knowing, however, that the semblance image in question is of a certain class, could aid in the development of a computerized velocity picker that uses a semblance image from the same class picked by a knowledgeable operator as a guide. Following this line of thought, I develop a package for automated velocity picking based on the following 3 steps:

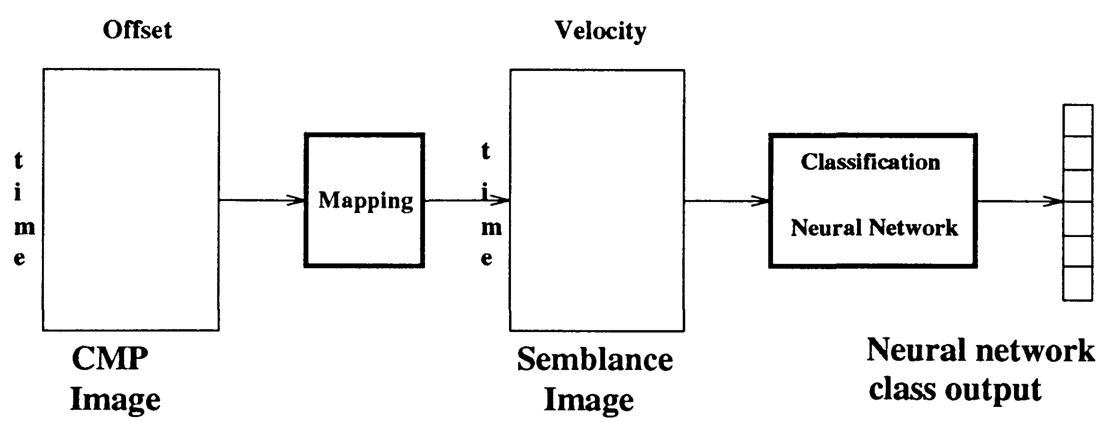


Figure 4.1. Overview of the classification processing: from CMP image to the classified output pattern.

1. A knowledgeable data processor examines a suite of semblance images from the data set in question, establishes semblance classes, and picks velocities from class-image representatives,
2. Using a neural network, semblance images are classified according to their pattern features,
3. Velocities are picked from the semblance images using the classification output as a guide to locate the correct velocity picks.

Semblance images are obtained from CMP gathers in the survey through a mapping process (see Figure 4.1). The mapping of a single point in a semblance image provides a measure of the signal coherence from a window of data being moveout hyperbolicly at a particular stacking velocity. The calculations defining this mapping process are detailed in appendix A. From Appendix A, it should be noted that several parameters must be chosen carefully. In particular: 1) the incremental velocity spacing should be in a constant $(\frac{1}{v_s^2})$, where v_s is the stacking velocity, 2) the time sampling should be 5 to 10 times larger than that used in the common-midpoint gathers, and 3) consideration should be given to the type of geology in selecting the trial velocity range. To reduce training time and for normalization reasons, the gray-scale semblance images are scaled between -0.5 and 0.5, and are then applied as inputs to the network either for training or for classification after the training phase has been completed.

The classification neural network is designed so that the number of output nodes equals the number of output classes. With the assumption that image classification depends directly on the different types of geology in the study area, I design the neural network so that the number of nodes displayed at the output equals the number of geologic type sections. This implies that the data processor can intelligently choose the number of geologic type sections in advance and select semblance images for the training set. Given that the data processor typically has access to a brute stack, from which a priori knowledge is available on the number, and the types of geologic sections involved, this is not, in general, a critical requirement. In the rare case where no such a priori knowledge is available, an alternative method would be to fix the number of image classes at some manageable level of 5 to 10. To generate semblance images from each class for learning and initial velocity picks, place a grid over the survey area, with the number of grid points equal to the number of classes defined. Semblance images are then constructed at the CMP locations associated with each grid point and assigned a class number.

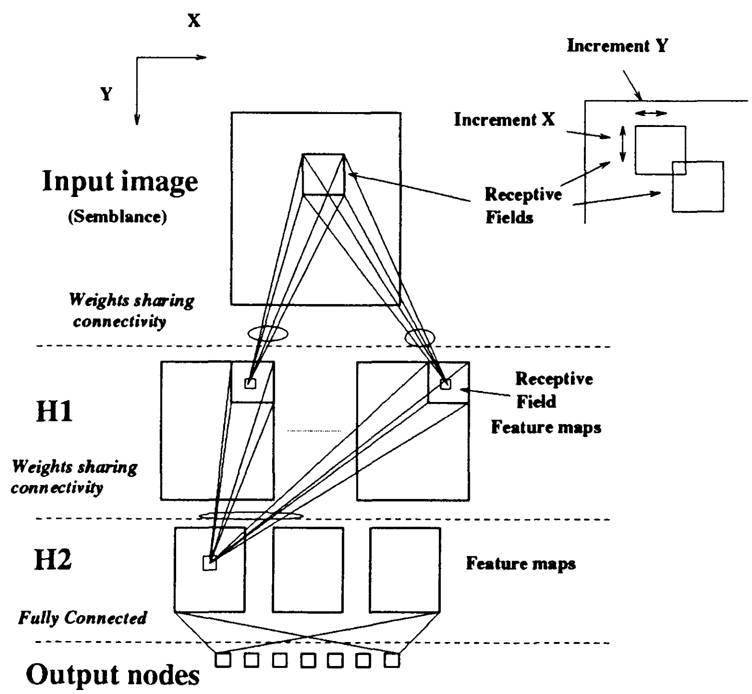


Figure 4.2. Two-layer weight-sharing, hierarchical neural network.

A weight-sharing network similar to the one described in section 3.3.2 is used to classify semblance images (see Figure 4.2). Determining the design criteria for such a network requires answers to the following questions:

1) What is the size of the receptive field (window) in the input image? The answer depends on the size of the characteristic features and on the sampling intervals along the time and sloth axis in the input image.
2) What are the X and Y-components of the increment between two-successive window positions? The answer depends on how disjoint the particular features are in the input image.
3) How many feature maps are in the first hidden layer (H1)? The answer depends on how complex the main features are in the input image.
4) What is the size of the receptive field (window) in the first hidden layer (H1)? The answer depends on how much resolution is needed to distinguish the main features in the input image.
5) How many feature maps are in the second hidden layer (H2)? This number should be greater than the number of feature maps in H1.
6) What are the increments between two-successive window positions in the first hidden layer? The answer also depends on the resolution of the main feature patterns in the input images.
7) How many hierarchical levels or hidden layers (e.g. H1 or H2) are there in the neural network? The number of hierarchical levels increases as the complexity of the problem.

Obviously, variations in the seven criteria produce an impossibly large number of alternative weight-sharing network designs. In order to reduce the number of possibilities in finding the optimum network, I use a priori knowledge in selecting these seven criteria. Examples of such a priori knowledge can come from: 1) previous experience in designing weight-sharing neural networks, or 2) observation of the input feature sizes in the input images to determine the size of the receptive field in the input image, or 3) the number of feature maps in H2 should be greater than the number of feature maps in H1, etc... Examples of such weight-sharing networks are given in Chapters 5, 6 and 7. However, even after using a priori knowledge, there is no way, to know whether or not the selected network architecture is optimum.

Once the architecture of the weight-sharing neural network has been designed, the network is trained to identify classes of input semblance images representing different geologies as interpreted from the seismic data. For every class, a set of semblance images are sampled from contiguous CMP's and collected into a training set. All images from this training set are run through the neural network until adjustment of the network parameters yields the expected class outputs according to a node-error learning criterion, thus, signifying the completion of the neural-network training process for recognizing these particular classes. Some care should be taken in choosing the node error bound. Training the neural network with an error bound that is too small (i.e. less than 0.05) should teach the network to precisely identify the image class patterns in the training set, but with the possibility of a poor generalization performance. On the other hand, choosing an error-bound that is too large (i.e. between .2 and .4) could lead to better generalization, but with an image class-pattern that is too imprecise.

I make two assumptions in using the trained neural network to classify unseen semblance images:

Assumption 1 For the classification of each untrained semblance image, the quality of the classification (i.e., how well the neural network classifies the image) is measured through the quality factor QF^1:

$$QF = 1 - \frac{\sum_i^{nclass} class_i - Max}{Max}, \tag{4.1}$$

where $class_i$ is *ith* output-node value, Max is the maximum output-node value, and $nclass$ is the number of classes. QF varies in the range $[1,\ 2 - nclass]$. I assume that below a certain user-defined threshold for the quality factor, the input images are considered a poor fit to the class represented by the output-node pattern[2], and are, therefore, *eliminated* from further processing. There are many other ways to construct a quality factor. The l-2 norm between the actual and the ideal output is an example. I choose the quality factor described in equation 4.1, because it has a floating minimum value depending on the number of classes. For a given threshold value, the severity of the test for acceptance of the classified images increases as the number of classes increases. As the number of classes increases, the possibility of arranging numbers in each class node also increases, and, therefore, the test for acceptance should be more severe. Using a l-2 norm for the quality factor will force, instead, in having a floating threshold as the number of classes changes, which is not the case when using the quality factor of equation 4.1.

Assumption 2 Consider two successive training sets for classes n and $n+1$. The semblance images generated from CMPs located between the CMPs used to define classes n and $n+1$, are assumed to belong either to class n or class $n+1$ and to no other class. If they are, however, classified in a class other than n

[1]The ideal classified output occurs when the maximum node amplitude equals one and is surrounded by nodes of zero value.

[2]The class of the input image is determined by the location of the node of maximum intensity at the output of the neural network.

and $n+1$, they are *eliminated* from further processing.

Once the class to which each semblance image belongs has been determined, the velocities are automatically picked from these images. Picking velocities from semblance images is performed *sequentially* on the classified data. I reject those images having a quality factor that falls below the user-defined threshold according to Assumption 1. I also reject those images that do not belong to the expected classes in accordance to Assumption 2.

In order to illustrate the automated velocity picking scheme from semblance images using the classification outputs, consider the example shown in Figure 4.3. There are 11 training semblance images for each of three classes: images from CMPs

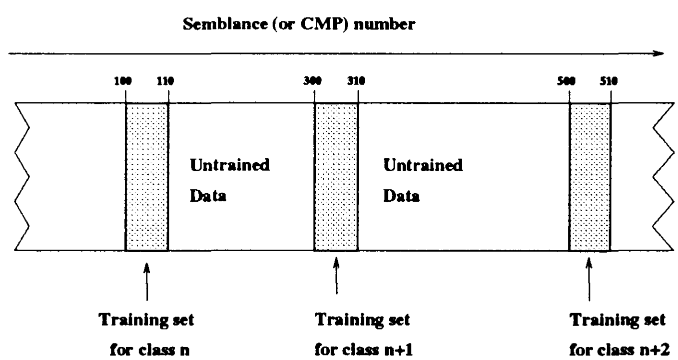


Figure 4.3. Location of training sets for different classes along the survey line .

100 to 110 represent the training set for class n, images from CMPs 300 to 310 represent the training set class $n+1$, and images from CMPs 500 to 510 represent the training set class $n+2$. Class n is characterized by three velocity picks, class $n+1$ is characterized by four velocity picks, and class $n+2$ is characterized by five velocity picks, indicating a change of geology along the survey line. Consider semblance images that have been classified from CMPs located between those used

to define the classes n and $n+1$. From Assumption 2, these images belong either to class n or class $n+1$. Starting from CMP 111 to CMP 299, assume the first CMP tested belonging to class n is CMP 111. The velocity picks from semblance image of CMP 110 are entered into the semblance image of CMP 111 as "first estimate", for locating the actual velocity picks in semblance image of CMP 111. User-defined, nonoverlapping rectangular areas are centered around the a priori picks in CMP 111. In each of the rectangular areas, we search for the maximum image amplitude. The sizes of the windows depends on the expected sizes of the input image features. Once found, these maxima, then, constitute the actual or a posteriori velocity picks for the semblance image derived from CMP 111. These a posteriori picks now become the a priori picks for the next semblance image belonging to class n. This process is repeated until the end of the untrained data at CMP 299 is reached. The same procedure is repeated for class $n+1$, but going in the reverse direction from CMP 300 to CMP 111. I continue the same process for untrained data between class $n+1$ and class $n+2$, then for classes $n+2$ and class $n+3$, and so on. After processing all the CMPs, I use the picked velocities to generate a velocity field over the entire survey area.

Because the process of velocity picking described above produces dense velocity sampling and because small, rapid variations in velocity create visible disturbances in the final stacked section, it is advisable to smooth the velocity field after the velocities have been picked using the computer technique previously described. In my research, I spatially smooth the velocity field using a 10-trace moving window average method for the sloth values.

Chapter 5

MARATHON MODEL DATA

In this Chapter, I apply the automated velocity picking scheme described in Chapter 4 to a set of data called the "Marathon data". The Marathon data set was obtained from a physical scale model representing a salt ridge over rifted basement as shown in Figure 5.1. The data were collected at the Seismic Acoustic Laboratory

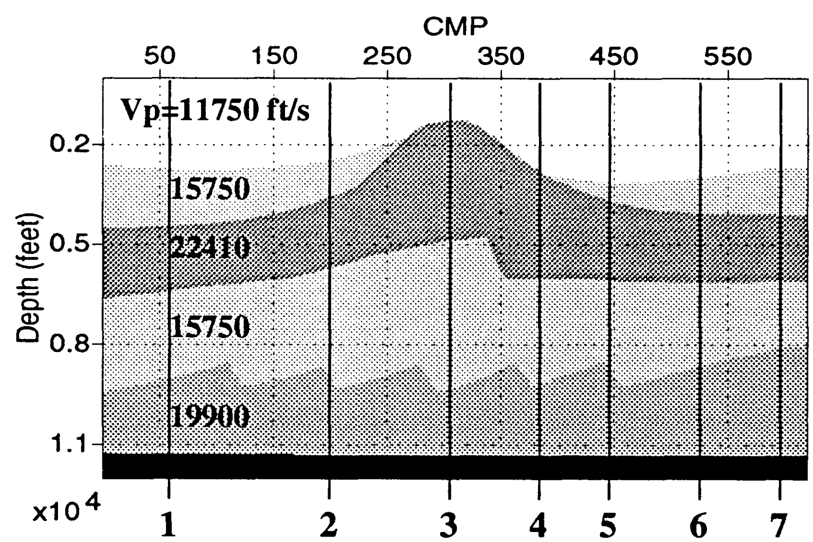


Figure 5.1. Marathon model showing the layered velocity distribution. The numbers 1, 2, 3, 4, 5, 6, 7 on the bottom of the Figure represent the locations of the training sets for each class.

at the University of Houston for the Marathon Oil Company. The end-of-spread acquisition configuration for these data is shown in Figure 5.2, and the recording

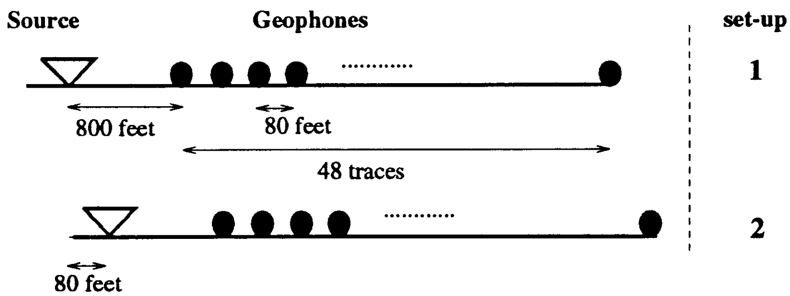


Figure 5.2. Two consecutive shot-geophone configurations (setups 1 and 2) for the Marathon data.

geometry is summarized in the Table 5.1. Examples of shot records and common

Table 5.1. Marathon field data parameters.

Number of shots	296
Number of channels/shot	48
Geophone spacing	80 feet
Shot spacing	80 feet
Near offset	800 feet
Far offset	4560 feet
Sample interval	4 ms
Trace length	2 s

midpoint gathers along the survey line are shown in Figure 5.3, demonstrating that shot records and CMP gathers look different when covering the different geologic sections represented in the model (Figure 5.1). In particular, different types of seismic events are noticeable such as a non-hyperbolic reflection on shot 100 at 1.5 sec., a hyperbolic reflection on shot 175 at 0.4 sec., and a diffraction event on shot 175 at 1.5 sec.

A total of 627 semblance images are generated from all CMPs in the data set

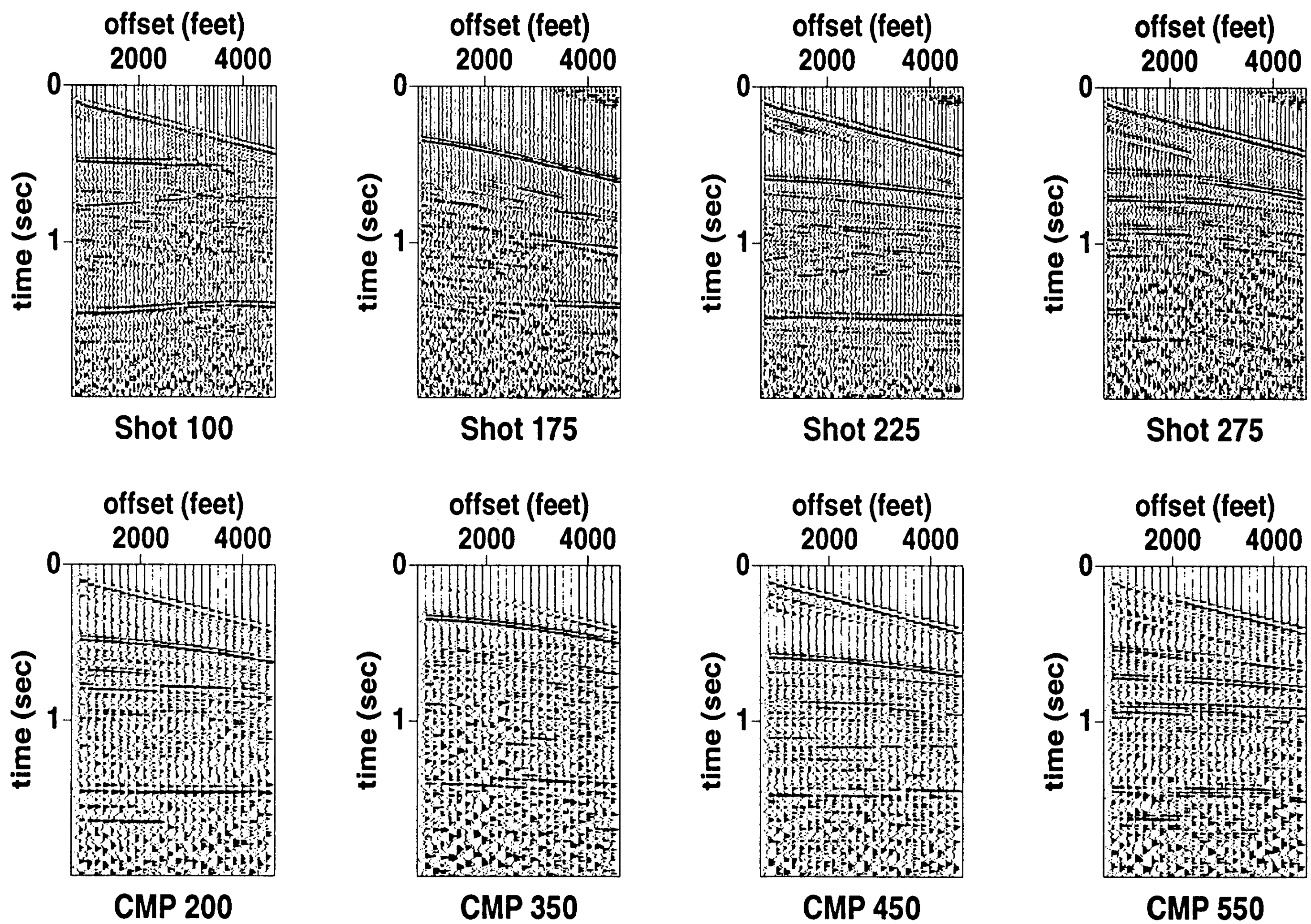


Figure 5.3. Examples of shot records and CMP gathers derived from the Marathon data set.

and the images are presented as input to the classification neural network. From the four samples of the computed semblance images (Figure 5.4), notice the different geometrical feature patterns with each image coming from a different part of the model (Figure 5.1). These patterns consists of a succession of peaks that become more elongated as the time increases, and the locations of the patterns along the *sloth* axis tend to decrease as the time increases. Some of these feature patterns have very steep peaks as in the semblance from CMP 550, or tends to have more gentle peaks as for semblance from CMP 450.

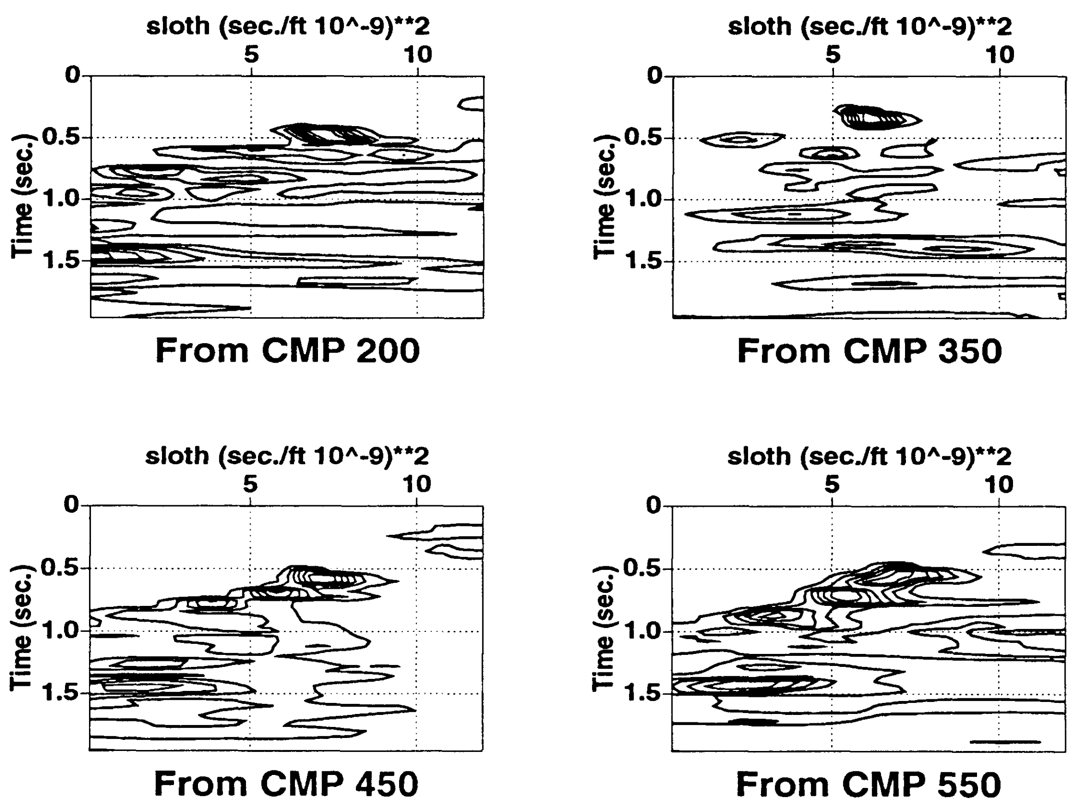


Figure 5.4. Examples of semblance images derived from the Marathon data set.

The number of data pixels in the semblance images depends on the time-sample interval[1] and on the number of trial *sloths*. The time-sample interval is chosen to be 40 ms, and with a 2-second trace length, there are 50 samples along the time axis of the semblance image. The *sloth* axis is sampled in constant $\Delta\frac{1}{v^2}$ increments and contains 60 *sloths* corresponding to 60 velocities in the range between 9000 and 45000 ft/s. Thus, the number of image-data pixels (representing the size of the input semblance image) is 3000 pixels (namely, 50 x 60).

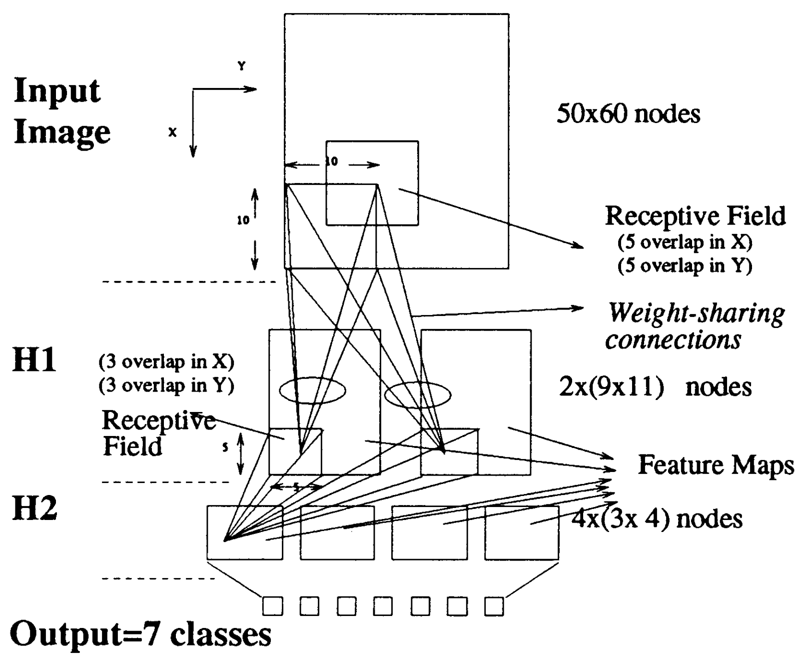


Figure 5.5. Architecture for the "Marathon-data" network used for semblance-image classification.

The local features in the input image are defined as peaks more or less elongated along the *sloth* axis (Figure 5.4). I decide that there are two types of elongated

[1]Time-sample interval: the interval between two consecutive values along the time axis.

Table 5.2. Architectural parameter description for the weight-sharing partially-connected network.

Description	Parameter Values
Input image size	60 x 50
Receptive field size	10 x 10
Overlap of the input receptive field	5 x 5
Number of feature maps in H1	2
Size of the feature map in H1	9 x 11
Number of feature maps in H2	4
Size of the receptive field in H1	2 x 5 x 5
Overlap of the receptive fields in H1	3 x 3
Number of output classes	7
Number of connections	22789
Number of free parameters (computed)	989
Type of connections Input-H1	weight-sharing
Type of connections H1-H2	weight-sharing
type of connections H2-Output	fully connected

features: one type for peaks which are more or less circular, the other type for peaks elongated along the *sloth* axis. In order to classify semblance images according to their feature patterns, a multilayer weight-sharing neural network with two feature maps in the first hidden layer is used to detect those two features. In order to determine the number of output nodes in the classification network, I interpret seven types of geology at the following locations in the model (see Figure 5.1): the left side of the model (1), the side right of the model (7), the salt dome (3), the left side of the salt dome (2), the right side of the salt dome (5), and two transition-zone types (2) and (4). I, thus, design the network with seven output nodes representing seven output-class patterns. The size of the window-receptive field in the input image is (10 x 10) pixels, which is the estimated size of the largest circular feature peaks. The overlap of two consecutive receptive fields in the input layer is either 5 pixels along the sloth axis or 5 pixels along the time axis. By overlapping five pixels instead of one pixel, some precision regarding the locations of the peaks is lost. Because a pattern class is not only defined by the locations but also by the geometrical shapes of the peaks, this partial loss of location information is not particularly significant. The size of the feature maps is, however, smaller than that of the input image. These two feature maps in the first hidden layer are scanned together four times by a window of size of (5 x 5) pixels, creating four high-order feature maps. The four feature maps are then fully connected to the seven-node output layer. The designed architecture has 22789 connections, but only 989 independent weights, demonstrating again the advantage in using a weight-sharing design that creates computationally powerful networks and that yields good generalization performance. The full architectural design is shown in Figure 5.5 and the design criteria are summarized in Table 5.2.

To train the neural network, I use a training set of 77 images (or 11 semblance images per class). Although the network has 989 free parameters, a training set of only 77 images is sufficient, because of the relative simplicity of the feature patterns inside the semblance image. Table 5.3 pinpoints the locations in the CMP domain, for the training set representatives of each class. Because the data have a high signal/noise ratio, and also because I defined too many output classes (as it will be shown later in this Chapter), a small training node-error learning criterion threshold of 0.05 was found to be sufficient to train the network. With this training node-error bound, I find that it takes 12,960 iterations or 168 passes (or epochs) through the entire training set to train the neural network successfully.

All the other semblance images that are not used in the training set (i.e. 627-77=550 images) are passed into the neural network for classification. The results of the classification and the corresponding quality factor describing the "goodness-of-fit" to a class are shown in Figure 5.6-a and Figure 5.6-b. As a first approximation, I assume that the classification is accurate if the quality factor is above 0.6[2], which corresponds to a normalized quantity of 0.93. Figure 5.6-b shows the quality factors for the classification of all semblance images along the survey line. Most of the quality factors are greater than 0.8 (or normalized 0.97) indicating that the classification performed by the neural network is accurate for most of the untrained images.

In analyzing the classification results (Figure 5.6-a), notice that the same semblance class is found in different locations in the survey. For example, semblance images classified in class 6, are located within the vicinity of training set samples for classes 6 and 1. The symmetry observed in the classification-CMP plot of Figure 5.6-a is directly related to the symmetry of the geology shown in Figure 5.1. This symmetry explains why the class-6 semblance images are positioned near the training-sample locations for classes 1 and 2 as well as 6 as shown in Figure 5.6-a. It therefore

[2]The maximum quality factor is 1 and the minimum quality factor is -5 in this case. If the range of values for the quality factor is scaled between 0 and 1, the value of 0.6 corresponds to a scaled value of 0.93.

Table 5.3. Representative sample images for training each class.

Semblance panels from CMPS	**Class number**
55 to 65	1
200 to 210	2
290 to 300	3
380 to 390	4
430 to 440	5
520 to 530	6
580 to 590	7

appears I have defined too many classes in characterizing the geology of the survey, and I would have obtained better classification results with fewer classes defined in more adequate locations.

Next, I apply the computerized velocity-search algorithm in order to pick the velocities automatically. During the automated search for time-velocity picks, I set the size of the search window to three samples along the time axis and five samples along the velocity axis[3]. Because the size of the search areas is set to half the size of the largest circular feature peak, which is roughly (3 x 5)-pixel samples in the input image and, therefore, the search areas usually contain a single peak. That prevents the limited area search algorithm from selecting incorrect peaks.

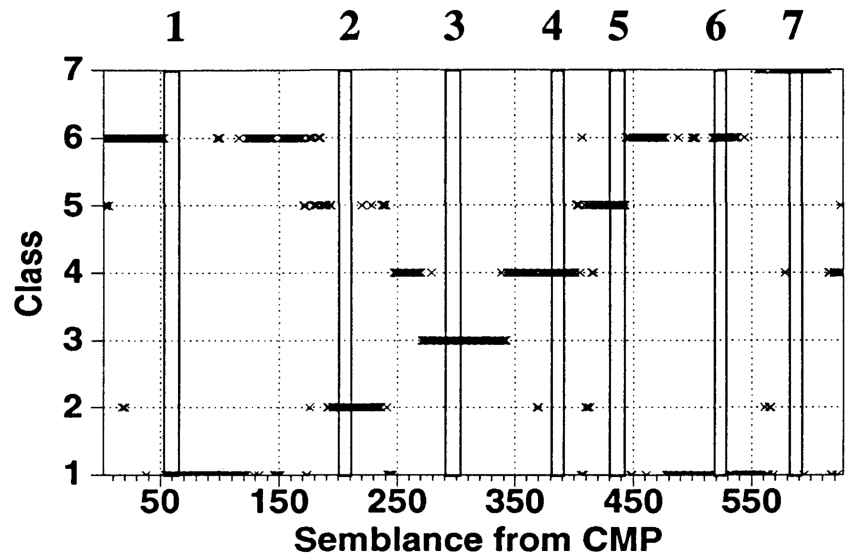
Using the suite of velocities picked by hand and those picked automatically by the velocity-search algorithm, velocity fields for each of the above cases are contoured in Figures 5.7-a and 5.7-b, respectively. Comparing these two fields, I quantify their similarity by computing the average percentage difference to be 3.8%, which is very good considering that the velocity is a robust parameter for the stacking process. Final stack sections are generated from these two velocity fields (Figures 5.8-a and 5.8-b). A visual comparison shows little difference between these two sections.

In order to gain insight on how to improve the selection of the network architecture criteria, I make a three-dimensional plot for each set of weights used to build the two feature maps in the first hidden layer H1 (weight amplitude versus the X and Y dimensions of the semblance image as shown in Figure 5.9). Notice that these plots are almost identical. Thus, concluding that the same operation is being unnecessarily performed twice, I redesign the network using only one set of weights to build the first hidden layer. The reduced architecture of this latest design now has a total of 388

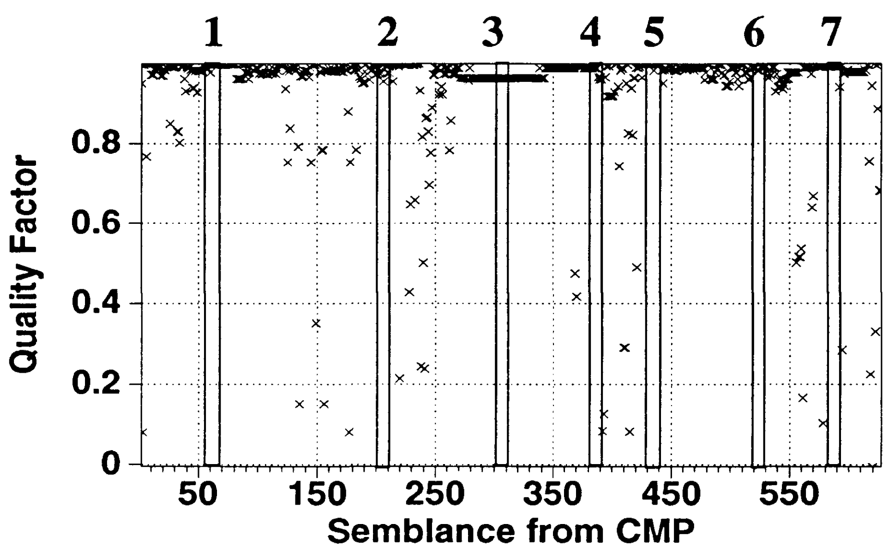
[3]Recall that each sample interval along the time axis represents 40 ms.

free parameters compared to the 989 in the original architecture. As demonstrated in Chapter 3, the new design with its reduced weight architecture should allow the network to exhibit a much improved generalization performance.

In conclusion, both techniques (the neural-network and the manual picking procedures) give equivalent results. In addition, the automatic scheme generates a continuous velocity analysis along the survey line. Such a continuous velocity analysis is needed when special processing techniques such as A.V.O. are used.



(a)



(b)

Figure 5.6. Classification results for the Marathon data set. The numbers 1, 2, 3, 4, 5, 6, 7 on the upper-horizontal CMP axis in Figures 6.6-(a) and 6.6-(b) represent the locations of the training sets for each class. Figure 6.6-(a) shows the output class to which each semblance image has been classified and Figure 6.6-(b) shows the corresponding image-class quality factor that is a measure of how well the image fits into its class.

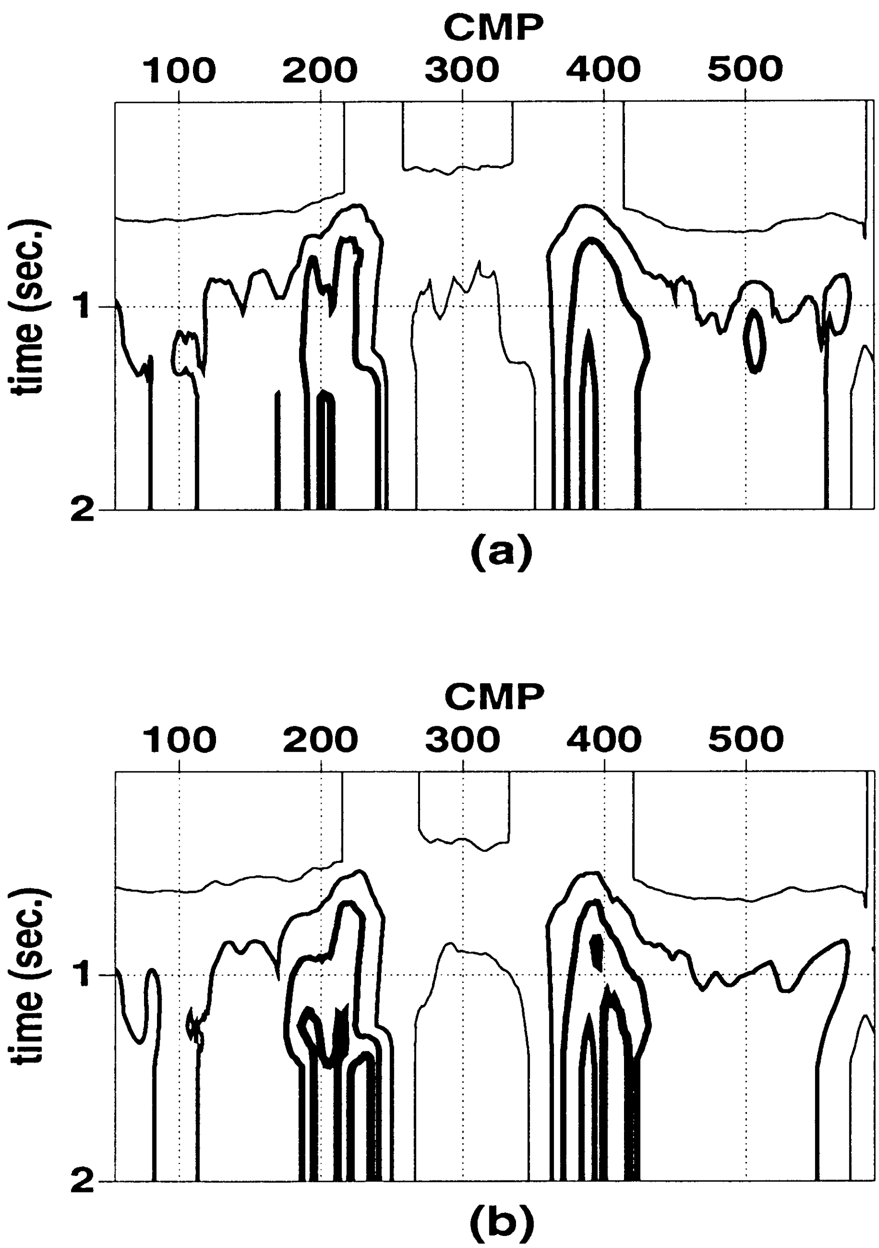


Figure 5.7. Contour plot of the stacking velocity field generated from velocities picked by hand (a) and that generated from velocities selected using the automated procedure (b). Contours were plotted at 12000, 17000, 27000 and 40000 ft/sec where the thinnest line represents 12000 ft/sec and the thickest line represents 40000 ft/sec.

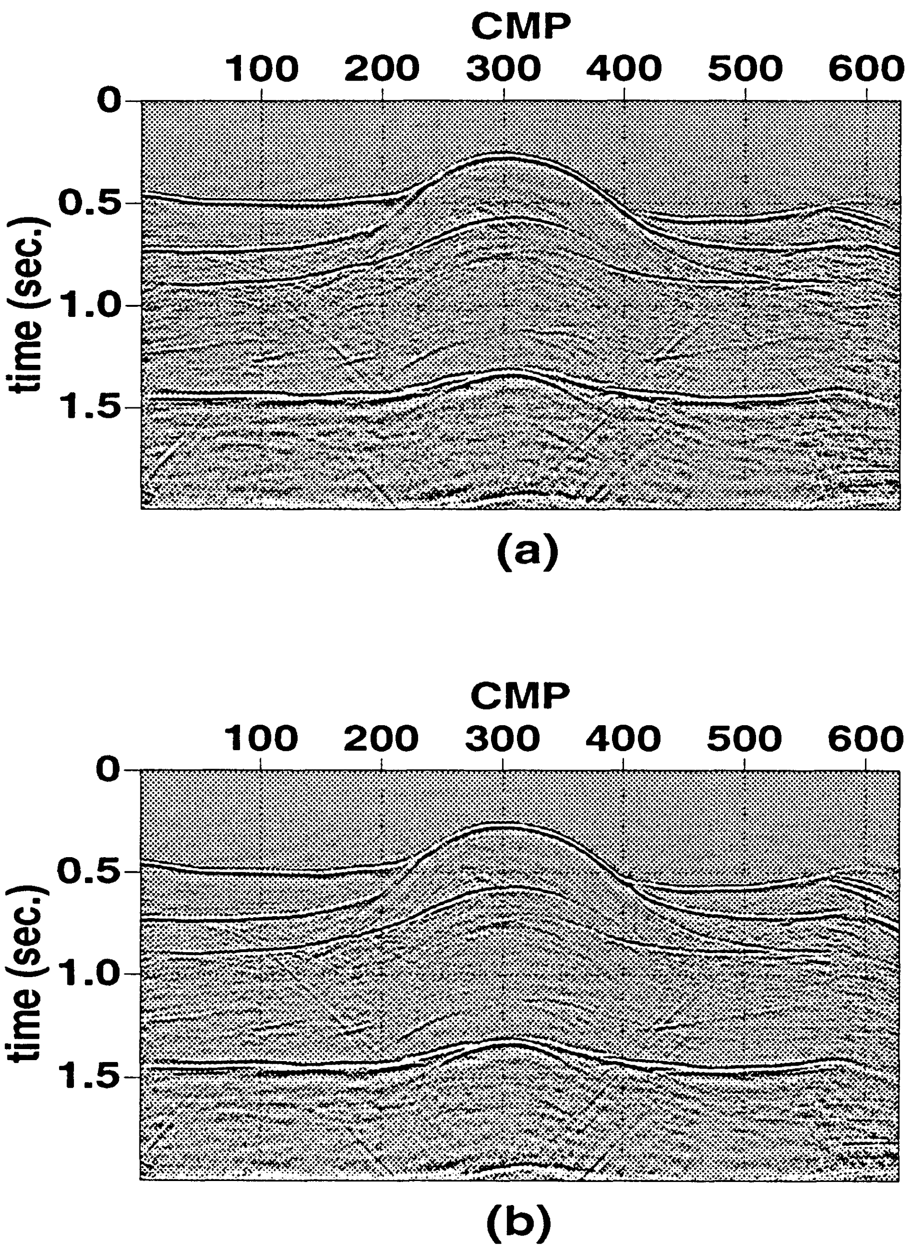


Figure 5.8. Comparison between the stack processed with velocities picked by hand (a) and the stack processed with velocities derived from the automated velocity picking procedure (b).

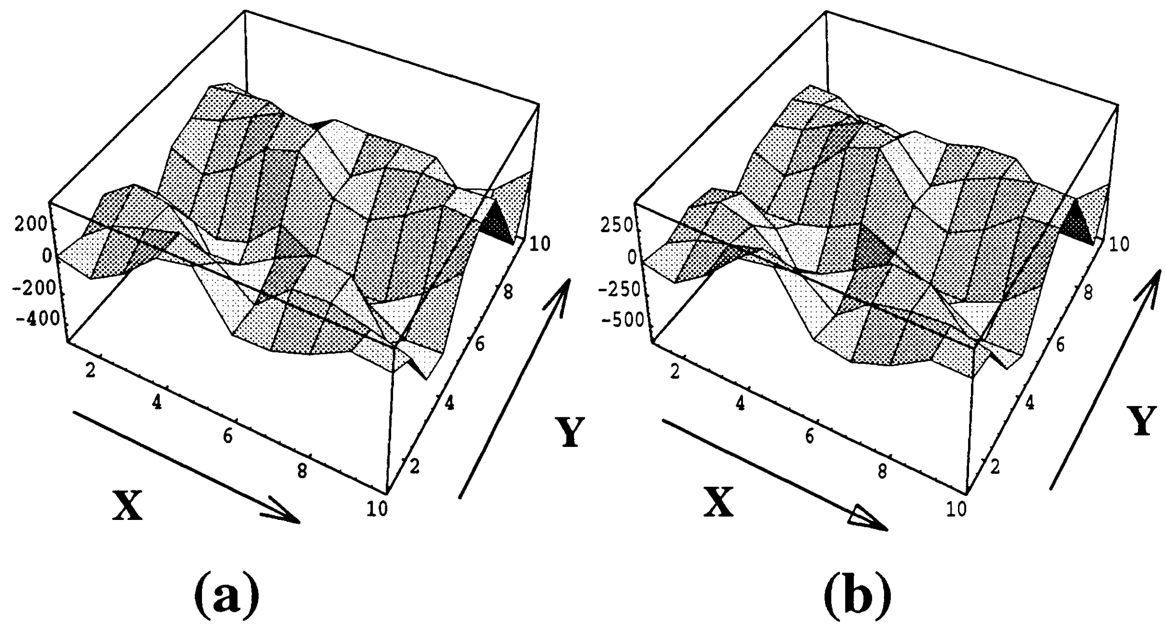


Figure 5.9. The two weight-sharing filters (a) and (b) from the input layer (semblance image) to the first hidden layer H1. The input image is convolved with each weight-sharing filter to construct the two feature maps in H1.

Chapter 6

COMPUTED RECORDS SECTIONS

In this Chapter, I apply the automated velocity picking technique to computer generated seismic records. Two hundred and fifty computed seismic records are computed using a ray-tracing software package[1], applied to the geological model shown in Figure 6.1. The recording geometry is summarized in Table 6.1.

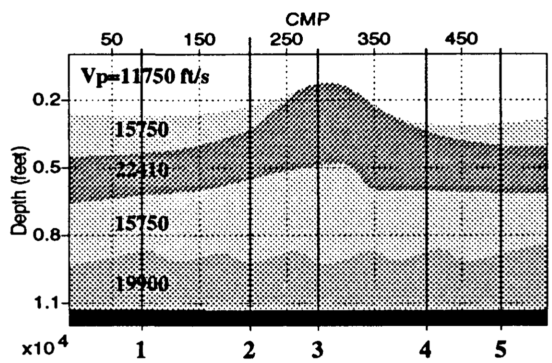


Figure 6.1. Synthetic-data model showing the layered velocity distribution.

Examples of generated shot records and common-midpoint gathers along the survey-line are shown in Figure 6.2. Notice that the seismic events in the CMP gathers varies with the different geologic sections covered in the model. In particular, different types of seismic events are noticeable such as a non-hyperbolic reflection on shot 75 at 1.3 sec., a hyperbolic reflection on shot 125 at 0.3 sec, and a diffraction

[1]Cshot, the ray-tracing modeling package, is a Colorado School of Mines software developed by the Center of Wave Phenomena.

Table 6.1. Synthetic field data parameters.

Number of shots	250
Number of channels/shot	48
Geophone spacing	80 feet
Shot spacing	80 feet
Near offset	800 feet
Far offset	4560 feet
Sample interval	4 ms
Trace length	2 s
Field configuration	One sided
CMP spacing	40 feet
Number of CMPs	546

event on shot 175 at 1.2 sec. Notice also that the reflections look more hyperbolic in the CMP domain than in the shot domain. (see Figure 6.2).

A total of 546 semblance images are generated from the CMPs in the data set. Examples of computed computed semblance images are shown in Figure 6.3, each exhibiting a different set of feature patterns. The feature patterns consist of a succession of peaks that become more elongated as the time increases. The procedure for determining the number of data pixels in the semblance images has already been described in Chapter 5. Following the same procedure, I calculate the total number of data pixels for each input semblance image to be 3000. After scaling the data pixels in the range between -0.5 and 0.5, the images are presented as inputs to the classification neural network.

To determeme the number of output nodes, examine Figure 6.1 and interpret five types of geology: the salt dome (3), the left side of the salt dome (1), the right side of the salt dome (5), and two transition zones (2) and (4). Corresponding to these five geologic classes, I construct a five-node network output. The design of the neural

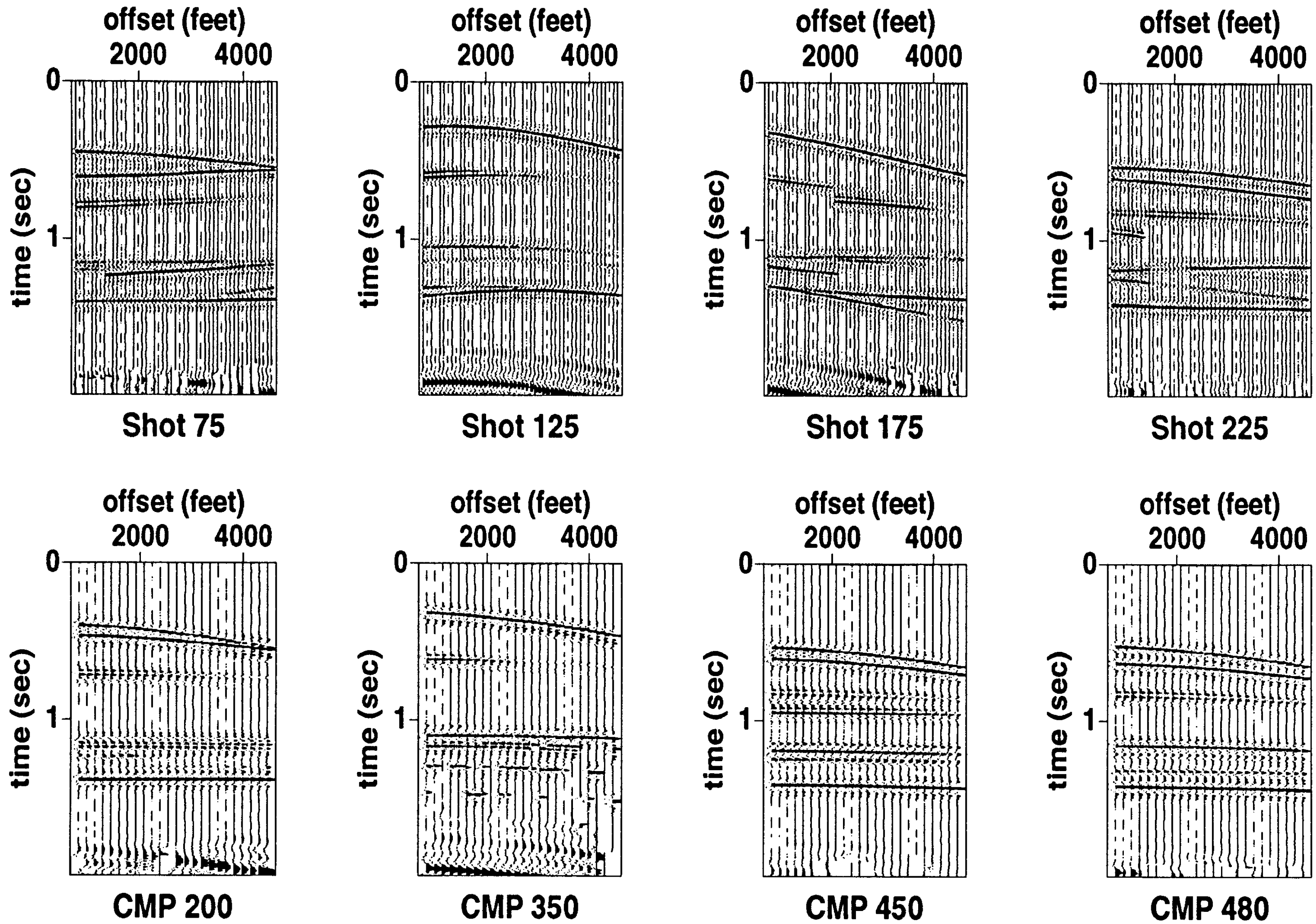


Figure 6.2. Examples of shot records and CMP gathers from the synthetic data set.

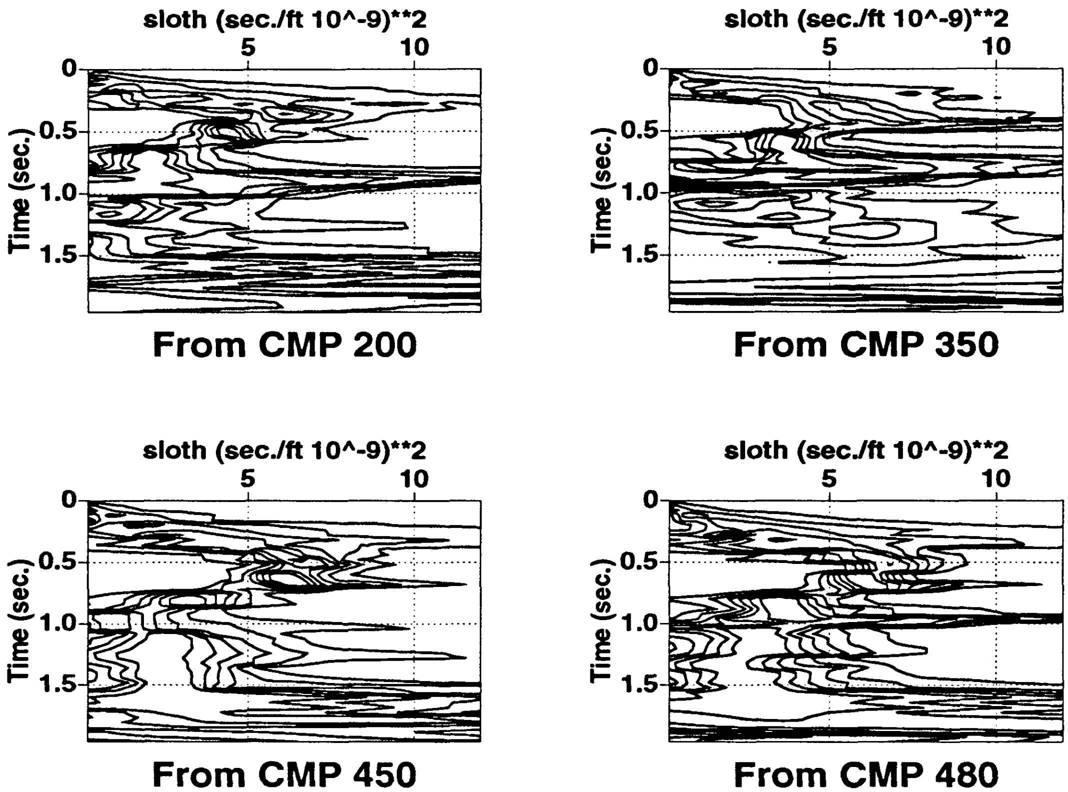


Figure 6.3. Examples of semblance images derived from the synthetic data set.

network is based on the characteristic features in the input semblance images, which are peaks elongated in the direction of the sloth axis as shown in Figure 6.3. Using the knowledge acquired from Chapter 5, I design a weight-sharing neural network with only one feature map in the first hidden layer and two feature maps in the second hidden layer. The second hidden layer is fully connected to the five output nodes. The sizes of the receptive fields of this network design are kept the same as those in Chapter 5. The designed architecture has 10753 connections but only 398 free parameters. A schematic of the architectural design for the classification neural

Table 6.2. Architectural parameter description for the weight-sharing neural network.

Description	Parameter Values
Input image size	60 x 50
Receptive field size	10 x 10
Overlap of the input receptive field	5 x 5
Number of feature maps in H1	1
Size of the feature map in H1	9 x 11
Number of feature maps in H2	2
Size of the receptive field in H1	5 x 5
Overlap of the receptive fields in H1	3 x 3
Number of output classes	5
Number of connections	10753
Number of free parameters	398
Type of connections Input-H1	weight-sharing
Type of connections H1-H2	weight-sharing
type of connections H2-Output	fully connected

network is shown in Figure 6.4. A summary of its design criteria are given in Table 6.2.

To train the network to recognize the five output classes, fifty-five semblance images (or 11 images per class) are used as the training set (see Table 6.3). Because of the high signal/noise ratio of the input, I choose a small training error node threshold of 0.05 in order to closely fit each training image to its class. With this training node-error bound, I find that it takes 1100 iterations or 20 epochs to train the neural network successfully.

All the other semblance images that are not used in the training set are passed into the neural network for classification. The results of the classification and the corresponding quality factors describing the "goodness-of-fit" to a class are shown in Figures 6.5-a and 6.5-b. The threshold quality factor for correctly identifying classified semblance images is set to 0.6 (or a normalized value of .90) Most of the quality factors are greater than 0.8 (or a normalized value of .95) indicating that the classification performed by the neural network is accurate for untrained images.

The velocity field for the geologic model of Figure 6.1 is derived using the automated computer search algorithm and is compared to the velocity field derived from the manual picking method (Figures 6.6-a 6.6-b). I find that the two velocity fields are visually similar, and furthermore, can be assumed to be nearly identical as indicated by the calculated average difference of 10% between the fields.

The NMO and stacking processes generated two sections, one using the velocities coming from the automated procedure and the other using the velocities manually picked in the semblance images. Upon close examination of the two stack-sections (Figure 6.7-a and Figure 6.7-b), no significant variations between the two sections was observed.

In attempting to improve on this design, I make a 3-dimensional plot of the single set of weights (weight-amplitude versus the X and Y dimensions of the semblance image as shown in Figure 6.8), and observe that the weight-filter structure exhibits symmetry in the Y direction. In essence, this symmetry indicates that each symmetric portion of the filter performs the same operation, an unnecessary feature of the present design. I can eliminate this undesirable effect by using a weight filter at one-half the size of the original. The result should yield fewer free parameters in the network architecture, and produce a corresponding improvement in its generalization performance.

Because both techniques (the neural-network and the manual picking procedure) give equivalent results, I conclude that the automated computer search algorithm is a valid technique for determining seismic stacking velocities for this model. This automated procedure is indeed, rather economical when applied to continuous velocity analysis as in A.V.O..

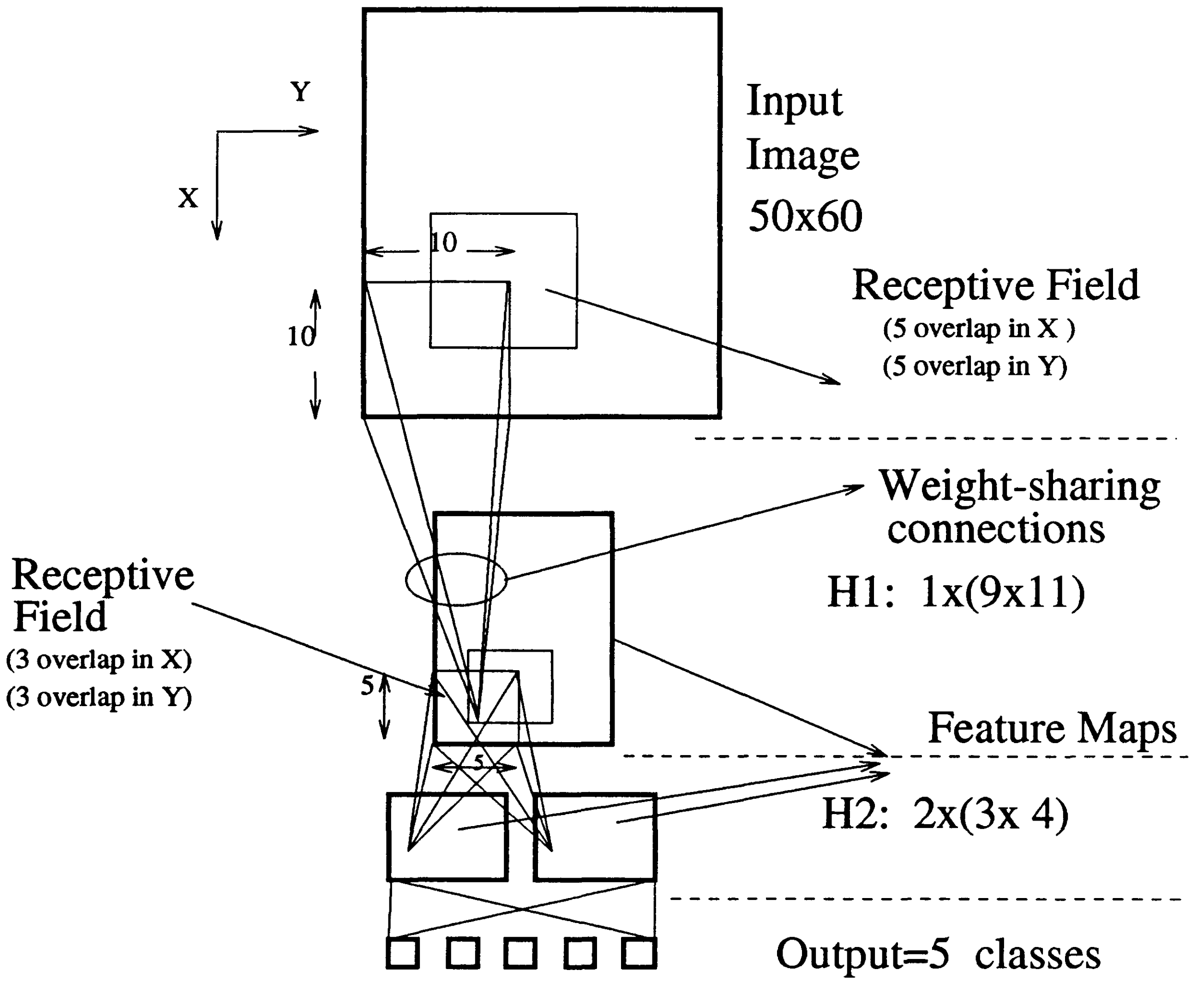


Figure 6.4. Architecture for the "synthetic-data" network used for semblance-image classification.

Table 6.3. Representative sample images for training each class.

Semblance panels from CMPS	Class number
85 to 95	1
200 to 210	2
290 to 300	3
410 to 420	4
490 to 500	5

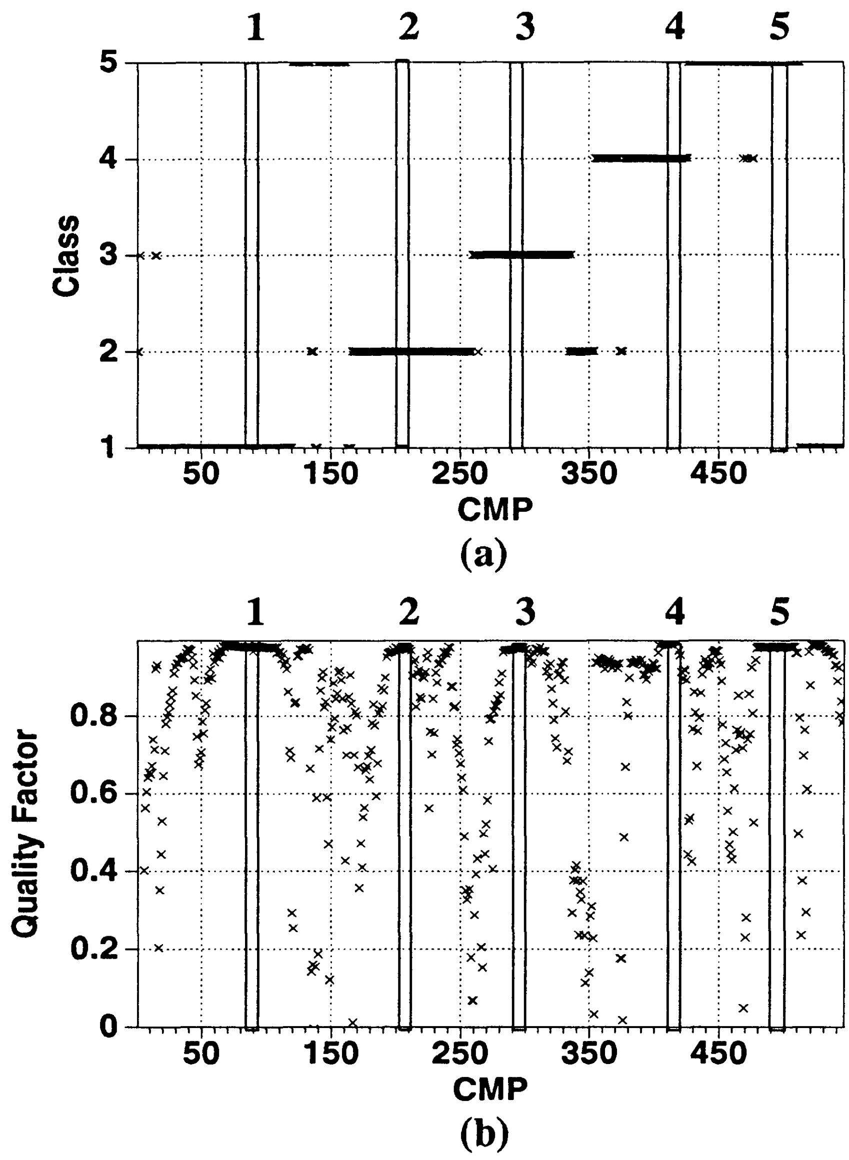
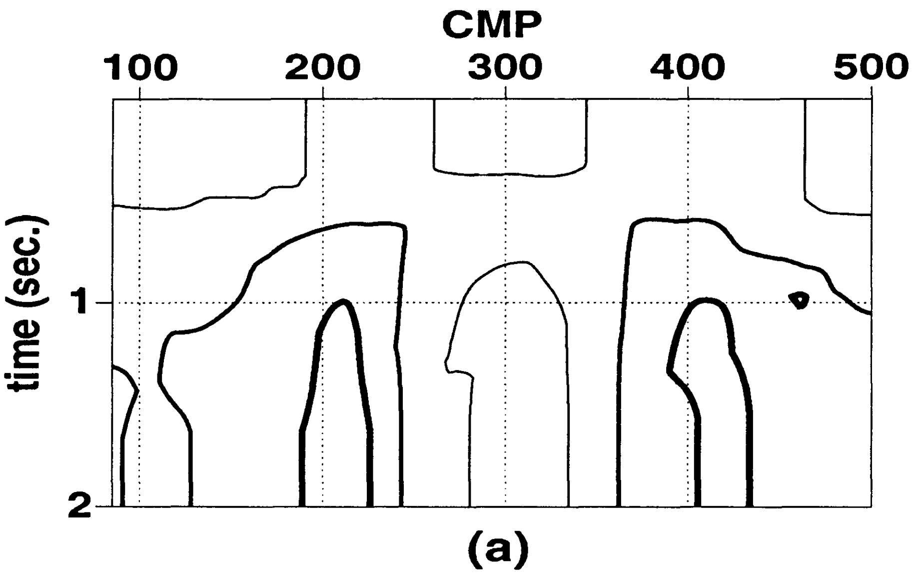


Figure 6.5. Classification results for the synthetic data set. The numbers 1, 2, 3, 4, 5 on the upper-horizontal CMP axis in Figures 7.5-(a) and 7.5-(b) represent the locations of the training sets for each class. Figure 7.5-(a) shows the output class to which each semblance images has been classified and Figure 7.5-(b) shows the corresponding image-class quality factor which is a measure of how well the image fits into its class.



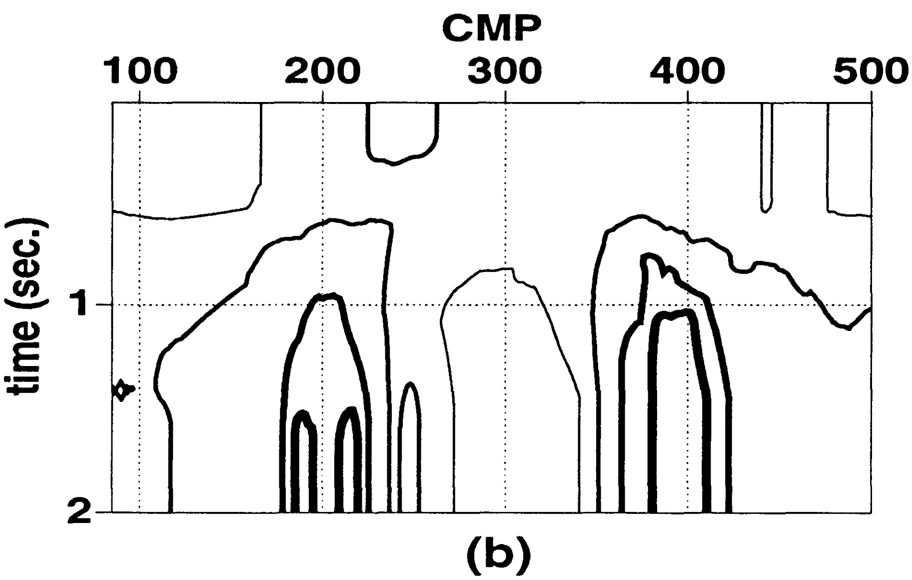
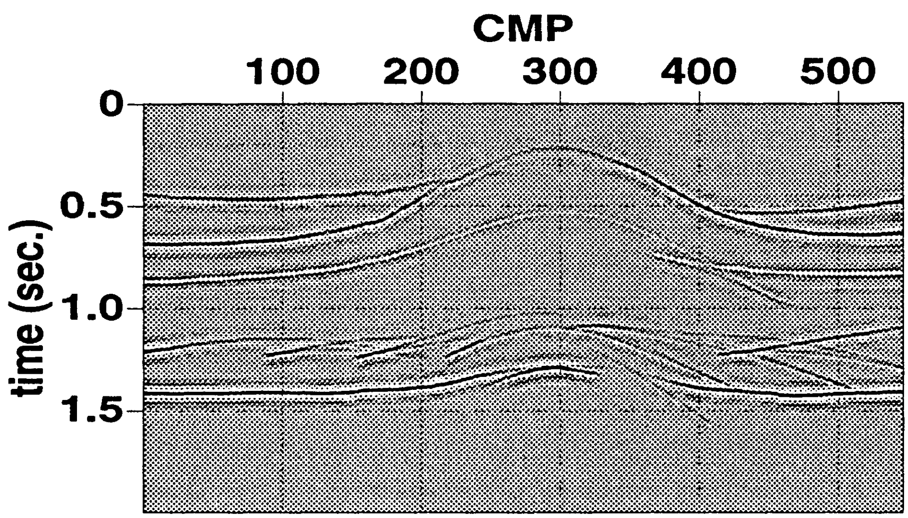
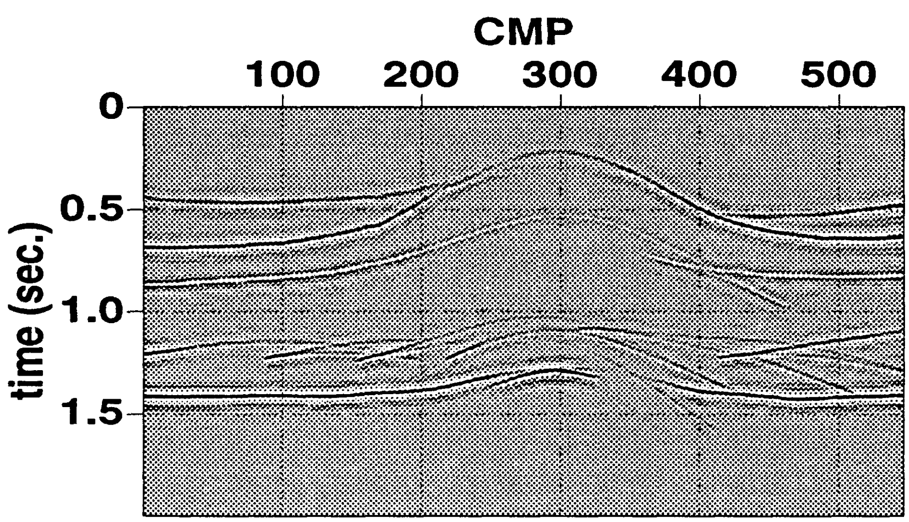


Figure 6.6. Contour plot of the stacking velocity field generated from velocities picked by hand (a) and that generated from velocities selected using the automated velocity picking procedure (b). Contours were plotted at 12000, 17000, 27000, 40000 ft/sec where the thinnest line represents 12000 ft/sec and the thickest line represents 40000 ft/sec.



(a)



(b)

Figure 6.7. Comparison between the stack processed with the velocities picked by hand (a) and the stack with the velocities derived from the automated velocity picking procedure (b).

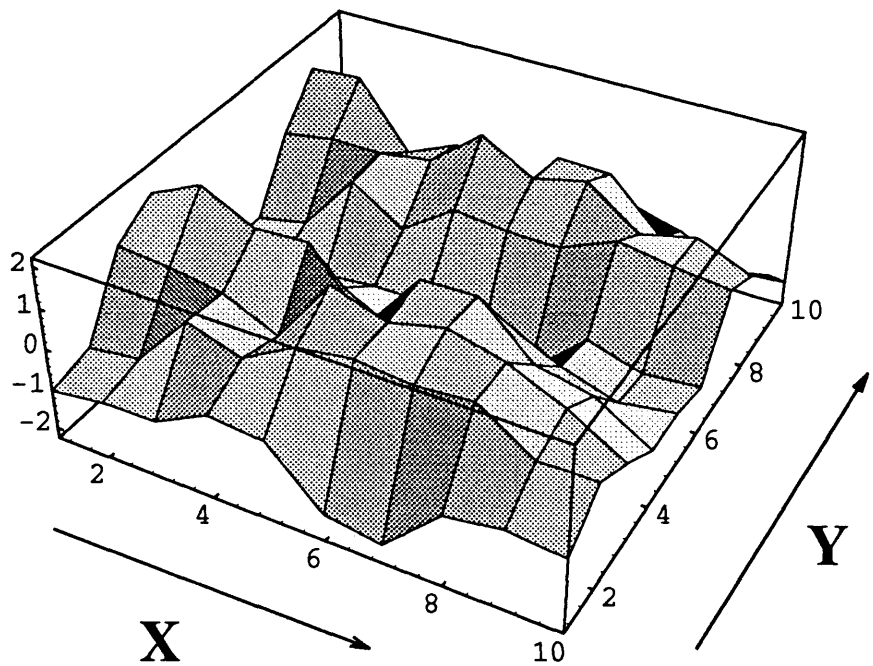


Figure 6.8. Weight-sharing filter from the input layer (semblance image) to the first hidden layer H1. The input image is convolved with the weight-sharing filter to construct a feature map in H1.

Chapter 7

FIELD DATA

In this chapter, I show the results of applying the automated velocity-picking technique to onshore field data. The field configuration to generate this data set is shown in Figure 7.1 and the recording geometry is summarized in Table 7.1.

Table 7.1. Real data field parameters.

Number of shots	117
Number of channels/shot	120
Field configuration	two-sided
Geophone spacing	55 ft
Shot spacing	220 ft
Near offsets (each side)	440/440 ft
Far offsets (each side)	4290/8690 ft
Sample interval	4 ms
Trace length	4 s
CMP spacing	27.5 feet
Number of CMPs	374
Maximum Fold	60

Examples of common-midpoint gathers and their corresponding semblance images along the survey-line are shown in Figure 7.2. The semblance images exhibit different feature patterns, demonstrating the presence of many types of images. The feature patterns are succession of peaks with varying degrees of inclination which are generally elongated along the *sloth* axis. The semblance images also look more noisy than those generated in the previous model studies (in Chapters 5 and 6).

A total of 374 semblance images are generated, one for each CMP in the survey.

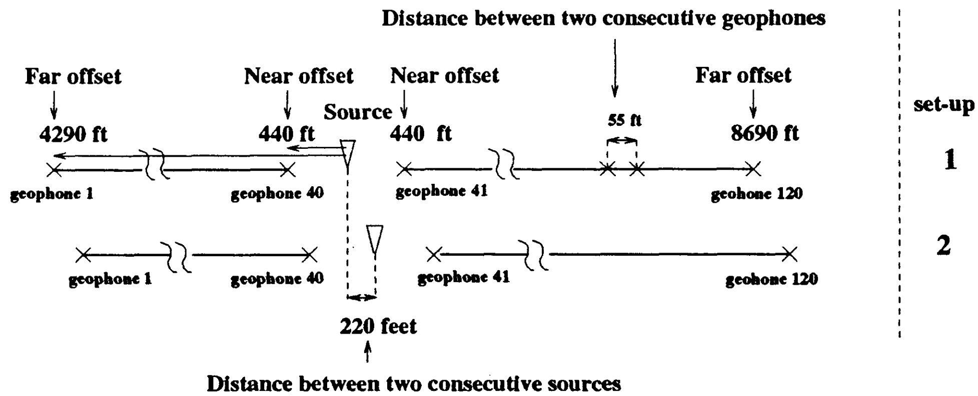


Figure 7.1. Two consecutive shot-geophone configurations for the field data set (setups 1 and 2).

These images are scaled in the range -0.5 to 0.5, and then they are presented as inputs to the classification neural network. From the 100 time values sampled along the vertical time axis and the 60 sloths sampled along the horizontal axis being in the velocity range between 8000 and 16000 ft/sec, the number of data pixels in the semblance image is, thus, 6000 (i.e, 100 x 60).

Using the three patterns interpreted from the field-data set, a neural network is designed to include a three-node output with two hidden layers H1 and H2 inserted between the input and the output layers. The schematic diagram of the network architecture and a summary of its design parameters are given in Figure 7.3 and Table 7.2 respectively.

To train the neural network to recognize the three output classes, a training set of 33 semblance images is assembled. Their locations in CMP numbers are given in Table 7.3. Because the signal/noise ratio of the semblance images for the field data are lower than those for the modeled data given in Chapters 5 and 6, I can expect the class image patterns to be less well defined for the field data. Therefore, I choose a

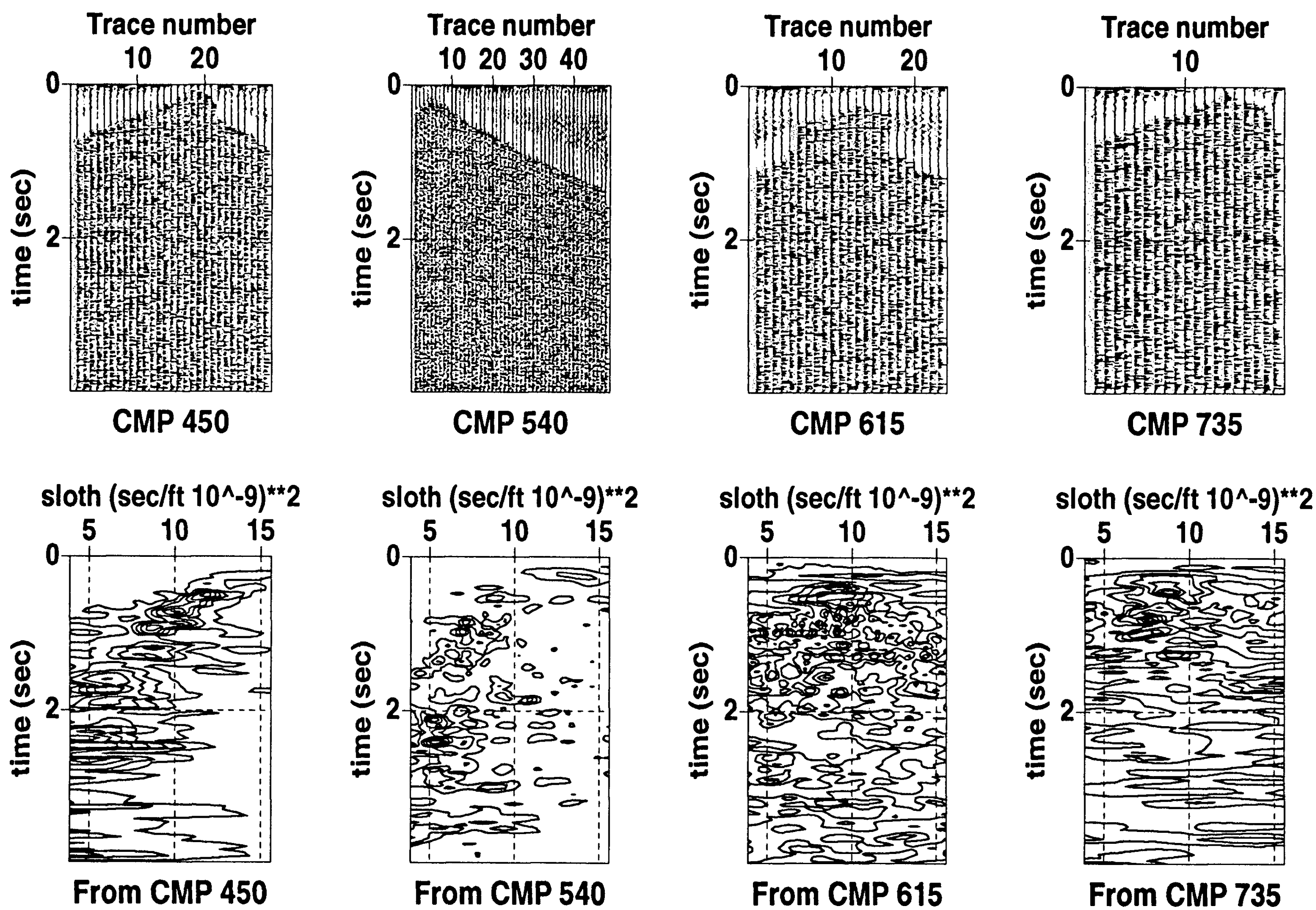


Figure 7.2. Examples of CMP gathers and their corresponding semblance images

more relaxed training node error threshold of 0.1. I find that it takes 1650 iterations or 50 epochs to train the neural network successfully.

All the other semblance images which are not in the training set are passed into the neural network for classification. The classification results and the corresponding quality factors describing the "goodness-of-fit" to a class are shown in Figure 7.4. Most of the quality factors are greater than 0.6[1] (or normalized quantity of 0.8) indicating that the classification performed by the neural network is fairly accurate for untrained images.

The velocity fields derived from velocities picked by the automated procedure and the velocities picked by hand are shown and compared in Figure 7.5-a and Figure 7.5-b. I find that the two velocity fields closely agree as indicated by the calculated average percentage difference of 6.3% between the two fields.

Two stacked-sections are generated, one using the velocities determined by the automated procedure and the other using the velocities manually picked in the semblance images. Because no significant variations can be detected between the two stack-sections (Figure 7.6-a and Figure 7.6-b), I conclude that the computer automated method for picking velocities is valid for this set of field data.

---

[1] A quality factor of 0.6 chosen as the minimum value for which the classification of the image is considered acceptable.

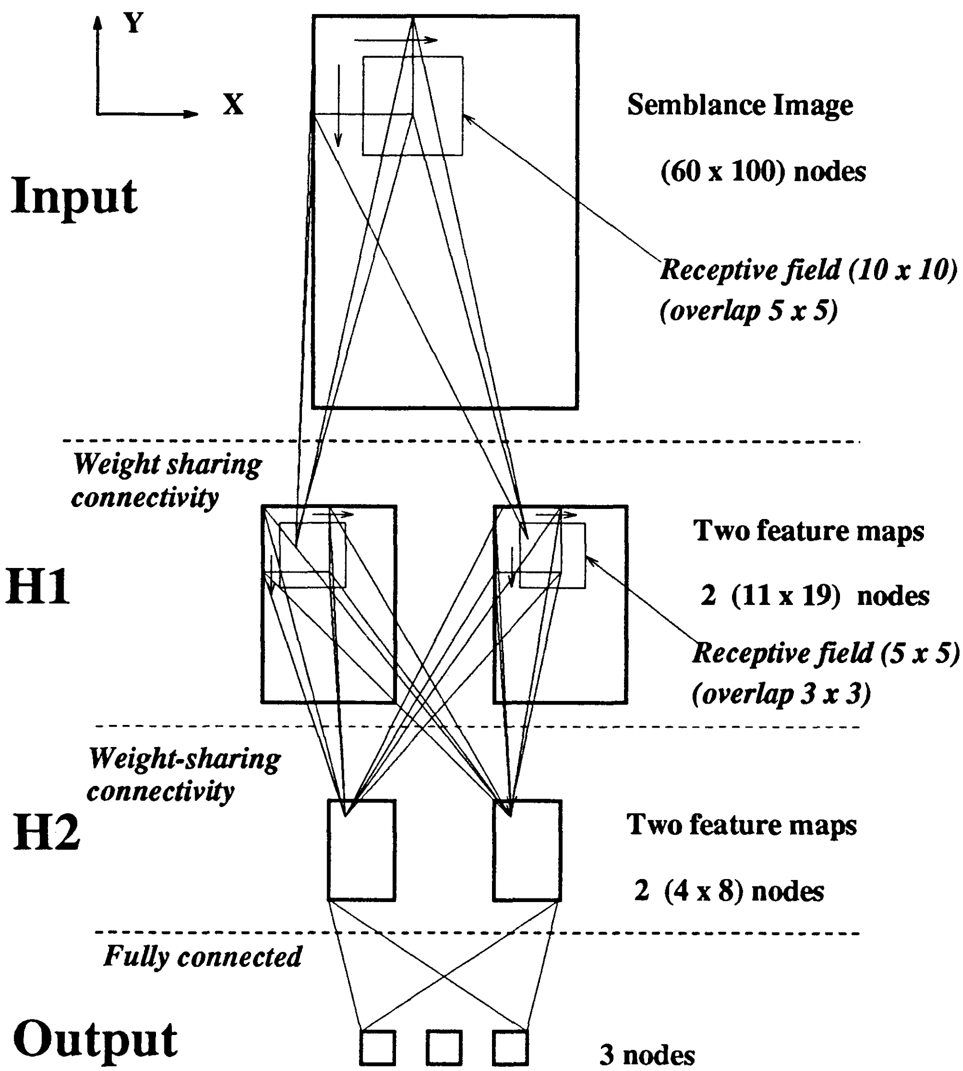


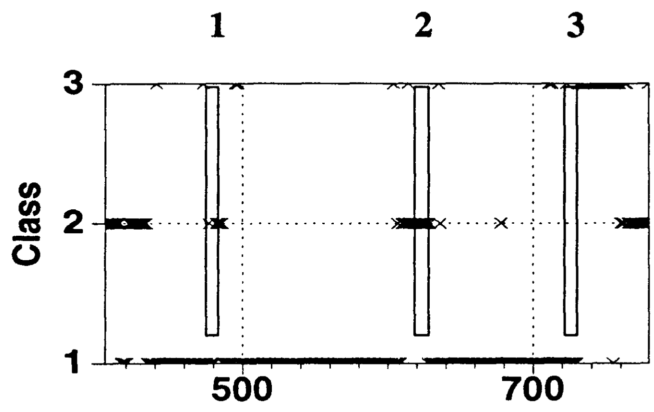
Figure 7.3. Architecture for the "field-data" network used for semblance-image classification.

Table 7.2. Architectural parameter description for the weight-sharing neural network.

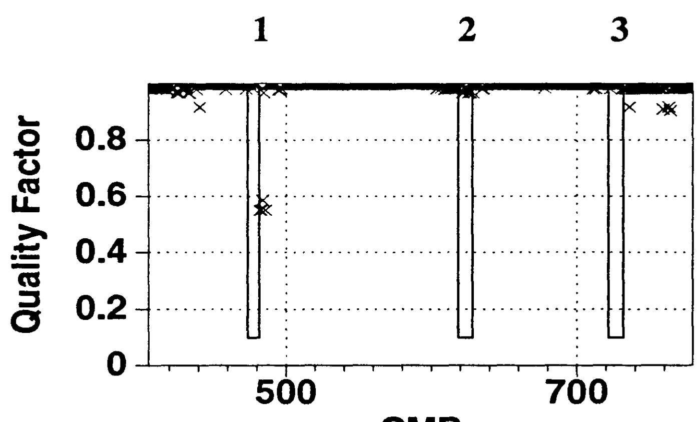
Description	Parameter Values
Input image size	60 x 100
Receptive field size in input image	10 x 10
Overlap of the input receptive field	5 x 5
Number of feature maps in H1	2
Size of the feature map in H1	11 x 19
Number of feature maps in H2	2
Size of the receptive field in H1	5 x 5
Overlap of the receptive fields in H1	3 x 3
Number of output classes	3
Number of connections	22374
Number of free parameters	977
Type of connections Input-H1	weight-sharing
Type of connections H1-H2	weight-sharing
type of connections H2-Output	fully connected

Table 7.3. Representative sample images for training each class.

Semblance panels from CMPs	Class number
450 to 460	1
615 to 625	2
725 to 735	3

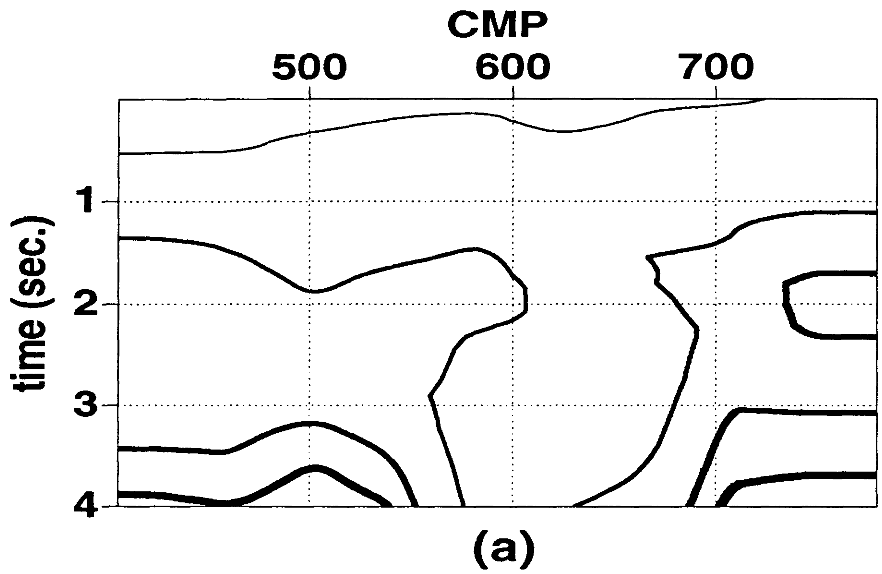


(a)



(b)

Figure 7.4. Classification results for the field data set. The numbers 1, 2, 3 on the upper horizontal CMP axis in Figures 8.4-(a) and 8.4-(b) represent the locations of the training sets for each class. Figure 8.4-(a) shows the output class to which each semblance image has been classified and Figure 8.4-(b) shows the corresponding image-quality factor which is a measure of how well the image fits into its class.



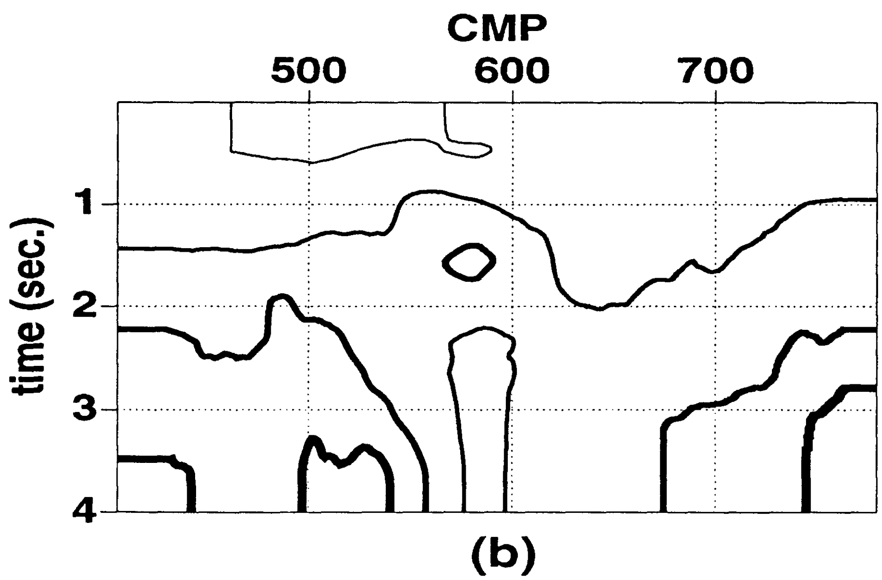
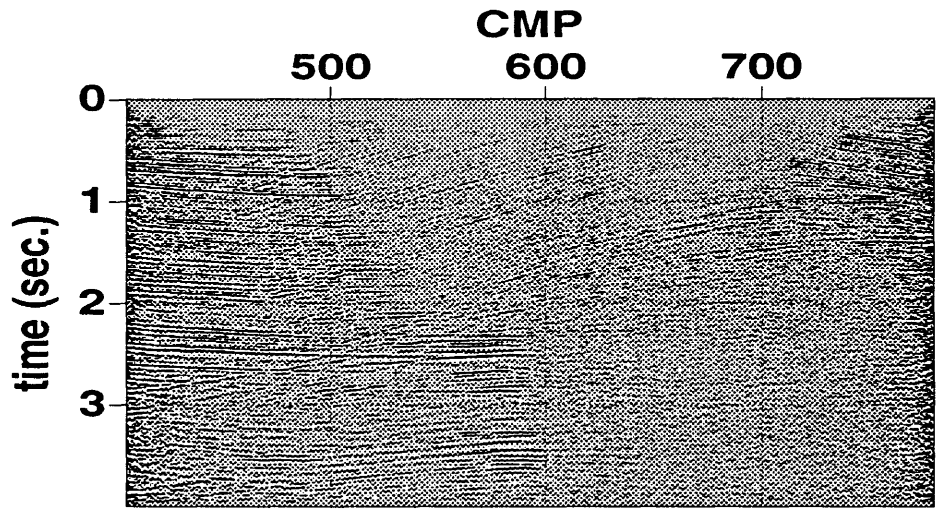
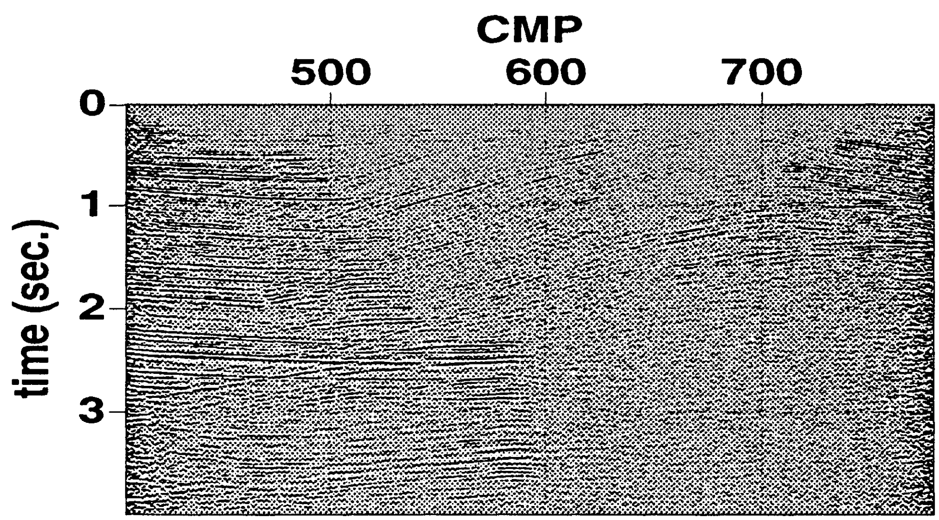


Figure 7.5. Contour plot of the stacking velocity field generated from velocities picked by hand (a) and that generated from velocities selected using the automated procedure (b). Contours were plotted at 9000, 12000, 14000 and 15000 m/sec where the thinnest line represents 9000 m/sec and the thickest line represents 15000 m/sec.



(a)



(b)

Figure 7.6. Comparison between the stack processed with velocities picked by hand (a) and the stack processed with velocities derived from the automated procedure (b).

Chapter 8

CONCLUSION

The purpose of this investigation has been to develop a computer-automated method to pick seismic velocities from semblance images. To achieve this goal, I designed a layered, hierarchical and constrained network architecture inspired by the mammalian neural system model (i.e that found in the retina of mammals), and applied this design to the classification of semblance images.

In the classical techniques of pattern recognition, the inputs to a classification system are normally feature vectors which contain high-level information. But instead, in this study, I use techniques to extract high-level information directly from low-level information data (i.e. images) presented as input to the classification system. Until now the technology to accomplish such a task has been time consuming and laborious, often requiring manual processing of the data. But my employment of neural-network technology demonstrates that unprocessed semblance-image input can be automatically and efficiently converted to higher-level data displayed as hidden-layer feature maps. Ultimately, the data which are displayed at the output of the neural network is a simplified pattern which defines class of the image.

A distinct advantage that neural networks have over the standard pattern-recognition techniques is that they can handle large amounts of low-level information in an efficient manner while automatically and accurately classifying the semblance image input. Because I tailor the neural network to the classification task by designing a constrained architecture (using weight-sharing connections and fixing the number and the size of feature maps in each hidden layer), I am able to process large amounts of low-level information input data.

Image classification works well when the differences between class patterns are highly variable, but may fail when changes become small and gradual. The way the neural network interpolates between classes is architecture dependent and difficult to predict. To bypass this problem, I compute a quality factor for the classification of each image, and eliminate those images showing low quality, thereby avoiding the interpolation issue altogether.

Semblance image classification is only a first step in achieving the goal of this study, the automated picking of seismic velocities. In the second and final step, the trained network output serves as a template to guide the computer in locating the correct velocity picks.

The automated procedure for picking seismic velocities was successfully tested on model data, synthetic data, and real data. The quality of the stacked sections generated by the computer was compared to that generated manually, and in all cases no obvious difference in the quality of the two equivalent sections could be found.

The above scheme of automated velocity picking is not really economical for conventional processing where only sparse locations of velocity analysis is used. However, when continuous velocity analysis such as in A.V.O., is needed, this technique becomes economical and viable because it automatically produces a velocity field densely sampled along the survey line.

Ideally it would be desirable to perform velocity picking in one step using a mapping neural network instead of the two-step technique inherent in the classification neural network. In such a system the input semblance image would be transformed by the neural network into an output image of the same size but with zeros everywhere except at the pick locations. Such a mapping network was tried, but the network did not generalize well.

Nevertheless, I have successfully introduced a new computer-automated technology, the classification neural network, for picking seismic velocities. It is my opinion that this technology of image classification is a powerful tool and may have significant applications in seismic data processing and interpretation.

Chapter 9

FUTURE DIRECTIONS

Although the classification neural network developed in this research project was successfully applied to the automated picking of seismic velocities, I believe this technology also has potential of being expanded to other geophysical applications through technique improvements in the design of the network architecture:

1. In the present method of architecture design, I must tinker with the design parameters on a trial basis, which at best is time consuming. Once an architecture is designed, I can never know if it is the optimum one without comparing it to other designs. Thus, I conclude that the present design methodology is impractical when trying to produce optimum networks. I recommend that future research be directed towards devising an automated and efficient design methodology for determining the optimum architecture in a reasonable amount of time.

2. In the application of automated semblance image classification to velocity picking, I did not have an image-size scaling problem because images of the same size are automatically created for a given survey. In addition, I was also fortunate in that local feature sizes do not change from one image to the next. But in other applications, image-size scaling may become a significant problem. For example, in the classification of seismic sections by geology, I will need to design a paradigm to scale all sections to the same size. But even this will not be enough since the same geologic feature may vary in size from section to

section.

3. The classification process can be easily extended to the 3-dimensional case where the input image, the feature maps and receptive fields are all 3-dimensional (e.g. one application is the classification of seismic data volume).

4. Finally another recommendation is the development of mapping neural networks since mapping is a common process used in seismic-data analysis, i.e. velocity and first-break picking. As I learned in the present study, it is not easy to construct mapping neural networks that generalize well. However, with the insight gained from the present work on classification neural networks, it may be possible to reduce the size of the architecture sufficiently and yield a mapping network design that will perform satisfactorily after training.

REFERENCES

[1] Abu-Mostafa, Y S., 1989, The Vapniks-Chervonenkis dimension: Information versus complexity: Neural Computation, **1**, 312-317.

[2] Baum, E. B., Haussler, D., 1989, What size net gives valid generalization?: Neural Computation, **1**, 151-160.

[3] Cover, T. M., 1965, Geometrical and statistical properties of systems of linear inequalities with application in pattern recognition: I.E.E.E. Trans. Elect. Comp. **14**, 326-334.

[4] Denker, J., et al., 1987, Large automatic learning, rule extraction, and generalization: Complex systems **1**, 877-922

[5] Elkington, A. E., 1991, The role of trace attributes in the implementation of neural networks for seismic first arrival evaluation: Master's thesis, Colorado School of Mines, **T-4084**.

[6] Falhlman, S. E., Lebiere, C., 1991, The cascade-correlation learning algorithm: CMU-CS-90-100, School of Computer Science, Carnegie Mellon University.

[7] Guyon, I., et al., 1992, Structural risk minimization for character recognition: Neural Information Processing Systems, **4**, Morgan Kaufmann Publishers.

[8] Hart, A., 1992, Using neural networks for classification tasks - some experiments on datasets and practical advice: J. Opl., Res. Soc. **43**, 215-226.

[9] Hertz, J., Krogh A. and Palmer P. G., 1991, Introduction to the theory of neural computation: Addison-Wesley.

[10] Hubral, P. and Krey, T., 1980, Interval velocities from seismic reflection time measurements: Society Exploration Geophysicists.

[11] Hubel, David H., 1988, Eye, Brain and Vision: Scientific American Library Series, **22**.

[12] Lane, D. F., 1989, Neural networks alternatives to seismic data processing: KEX Sponsors Report, Fall 1989, Colorado School of Mines.

[13] LeCun, Y., et al., 1989, Backpropagation applied to handwritten zip code recognition: Neural Computation **1**, 541-551.

[14] Marr, David, 1982, Vision: A computational investigation into the human representation and processing of visual information: San Francisco : W.H. Freeman.

[15] Martin, G. L., Pittman, J. A., Recognizing hand-printed letters and digits using backpropgation learning: Neural Computation **3**, 258-267, 1991.

[16] Mozer, Michael, 1991, The connectionist approach of artificial intelligence, Class packet, University of Colorado at Boulder.

[17] McCormack, M. D., 1990, Seismic trace editing and first break picking using neural networks: 60th Ann. Internat. Mtg., Soc. Expl. Geophys., Expanded Abstracts, 321-324.

[18] McCormack, M. D., 1991, Neural computing in geophysics: The Leading Edge of Exploration, January 1991, volume 10, number **1**.

[19] Ripley, B. D., 1992, Classification and clustering in spatial and image data: In Analyzing and Modeling Data and Knowledge, ed. M. Schader, Springer Verlag, pp. 93-105.

[20] Rummelhart, D. E., McClelland, J. L., 1986, Parallel distributed processing: Exploration in microstructures of cognition, vol. **1**: MIT Press.

[21] Rummelhart, D. E., Hinton G. E., Williams R. J., Learning representations by back-propagating errors: Nature, vol. **329**, October 1986.

[22] Schmidt, J., 1992, Neural network stacking velocity picking: PhD thesis, Colorado School of Mines, **T-4218**.

[23] Sheriff, R. E., 1988, Encyclopedic Dictionary of Exploration Geophysics: Society of Exploration Geophysicists.

[24] Neidell, N. S., Taner, M. T., 1971, Semblance and other coherency measures for multichannel data: Geophysics, **36**, 482-497.

[25] Tveter, D., 1991, Getting a fast break with backprop: AI expert, July 1991.

[26] Veezhinathan J., Wagner D., Ehlers J., 1990, First break picking using neural network. Preprint from Amoco Production Company, Tulsa Research Center, 4502, East 41st Street, Tulsa, OK - 74102.

[27] Wasserman, P.D. 1989, Neural Computing: Theory and Practice: Van Nostrand Reinhold.

[28] Wechsler, H., 1990, Computational vision: Academic Press Inc..

[29] Widrow, B., Lehr, M. A., 1990, 30 years of adaptative neural networks: perception, madaline and backpropagation: Proceedings of IEEE, vol. **78**, no. 9, September 1990.

[30] Yilmaz, O., 1987, Seismic Data Processing: Society of Exploration Geophysics.

Appendix A

SEMBLANCE IMAGES

It is assumed that four types of seismic events approximate hyperbolic traveltime curves: the **primary**, the **multiple**, the **diffraction** and the **conflicting-dip** events. In seismic data processing, only primary and conflicting-dip events need to be imaged. The multiple and diffraction events, however, might tend to hide the primary and conflicting-dip events, thus, making their detection more difficult. Another difficulty is that primary and conflicting-dip events cannot be imaged simultaneously.

Although primary events are assumed to follow hyperbolic paths, it is known that such paths are not exactly hyperbolic. For the single earth model, the assumption is true whether the interface is flat or has a constant dip. For cases involving more than one layer or a single layer case where the interface is curved, the assumption of hyperbolic moveout cannot be justified. However, when the subsurface layer geometry is moderately gentle, such as curved interfaces over large distances, the reflection patterns will closely approximate hyperbolic paths.

Diffraction and **backscattering energy** have higher velocities and water-bottom multiples have lower velocities compared to those of primary events. Because of these differences in velocities, the stacking process often acts as a velocity filter, partially suppressing these interfering events.

Conflicting-dip events are seismic events with different dips occurring at the same location at the same time. Each event in a conflicting-dip case, when processed separately, yields a different velocity, with the higher velocity belonging to the event with the greatest dip. In normal seismic processing, only one event can be processed

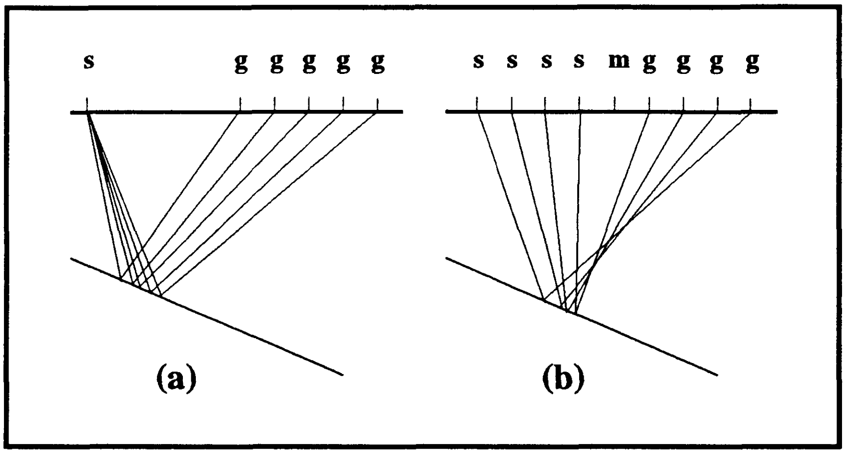


Figure A.1. Shot (a) and CMP (b) configurations (s: shots; g: geophones; m : midpoint).

at a time, thereby causing the loss of information contained in the other events. This creates a problem in processing conflicting-dip data, but can be resolved by using a specific process or technique called the **dip-moveout**.

A.1 Normal-Moveout and Stacking

Seismic data, although recorded in the shot-geophone domain, are mostly processed in common-midpoint domain (CMP). A common midpoint gather is defined as a set of traces which have been produced by a set-up of shots and geophones having the same midpoint (Figure A.1). Because the reflections look more hyperbolic in the CMP domain than in the shot domain, it is an advantage to process the seismic data in the CMP domain. Two main processing steps are applied in sequence in the CMP domain:

1. **Normal-moveout or NMO** that prepares the data for the stacking process by flattening the primary reflection paths,
2. **Stacking** that adds the data traces horizontally in a CMP gather considerably increasing the signal/noise ratio.

The equation which describes the hyperbolic path of the reflection traveltime can be written:

$$t^2(x) = t^2(0) + \frac{x^2}{v_s^2}, \tag{A.1}$$

where x is the offset or the distance between the source and the receiver positions, v_s is the stacking velocity and $t(0)$ is the traveltime when the offset is equal to zero. The stacking velocity v_s of a single reflection is derived from the hyperbola (equation A.1) that best fits that reflection over the entire spread length in the CMP gather. Normal-moveout correction at a given offset x is the difference between the two-way traveltime $t(x)$ and the zero-offset two-way traveltime $t(0)$ at $x = 0$:

$$\begin{aligned} \Delta t_{NMO}(x) &= t(x) - t(0) \\ \Delta t_{NMO}(x) &= t(0)\left(\left[1 + \left(\frac{x}{v_s t(0)}\right)^2\right]^{1/2} - 1\right), \end{aligned} \tag{A.2}$$

where $\Delta t_{NMO}(x)$ is the traveltime-correction used to flatten the reflection patterns in a CMP gather.

After completing the normal-moveout corrections, the stacking process is applied, horizontally combining seismic traces in a common midpoint gather. This process has the advantage of increasing the signal/noise ratio and compressing the amount of data. Compression of the data reduces the data from many offset traces in a CMP gather into a single "pseudo-zero offset" trace in the stack section.

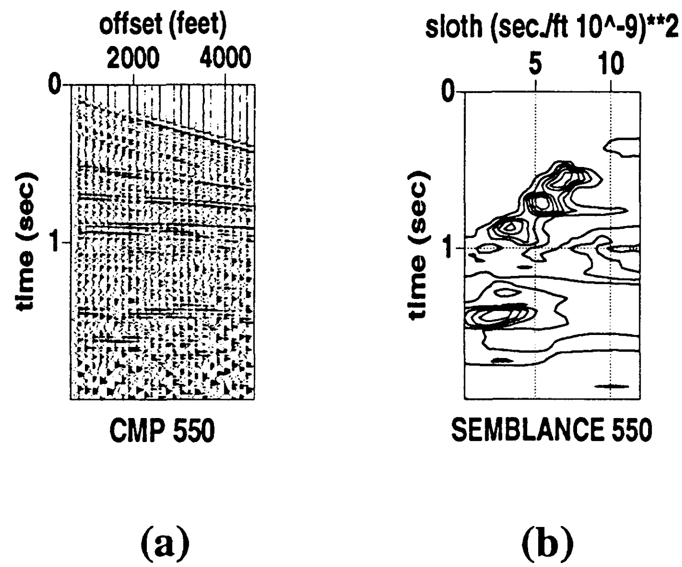


Figure A.2. Example of a CMP gather (a) mapped into a semblance image (b).

A.2 Semblance Images

Semblance images are used by the geophysicist in selecting seismic-stacking velocities. Semblance is a mapping from a time-offset domain (CMP domain) into a time-velocity domain (semblance domain). An example of mapping a CMP gather into a semblance image is shown in Figure A.2.

A.2.1 Semblance-image Computation

The process of creating a semblance image starts in the CMP gather domain. Different constant stacking velocity models are generated along time windows with each window being mapped into a single point in the semblance image. This mapping of a single point in the semblance domain provides a measure of the signal coherence that follows the reflection-signal hyperbolic path in the CMP domain.

In the CMP domain, the signal coherences are computed for a particular velocity by dividing the domain into non-overlapping incremental areas whose boundaries parallel the hyperbola corresponding to that particular constant velocity (equation A.1). These incremental areas are called time windows. The coherence signal amplitude following the hyperbolic path is computed for each time window according to the equation:

$$S = \frac{1}{M} \frac{\sum_{t(i)=ts}^{te} \left(\left(\sum_{i=1}^{M} f_{i,t(i)} \right)^2 \right)}{\sum_{t(i)=ts}^{te} \sum_{i=1}^{M} f_{i,t(i)}^2} \tag{A.3}$$

where $f_{i,t(i)}$ is the amplitude value on the ith trace at two-way time $t(i)$, and M is the number of traces in the CMP gather. The two-way time $t(i)$ lies along the trial stacking hyperbola. t_s is the starting time of the window and t_e is the ending time of the window.

A.2.2 Semblance-image Computation Parameters

In order to determine the best stacking velocities for the given data, these coherence computations are repeated (equation A.3) for several constant velocities. These computations constitute the mapping process which generates the semblance image. There are three conditions to consider when computing a semblance image:

1. The choice of the range of trial velocities needed to represent the data,
2. The choice of the spacing between trial velocities,
3. The choice in the number of time sample interval (or height) of the window where the coherence is computed.

Based on the geophysicist's past experience with different geologies and the velocities associated with these geologies, he should be able to select maximum and minimum velocities which define the trial velocity range. We should keep in mind that dipping events have anomalously high stacking velocities so that the geophysicist should choose an appropriately high maximum velocity, for this case.

A judicious choice of velocity spacing is a constant $\Delta\frac{1}{v_s^2}$, also called constant **sloth**. This velocity spacing follows from the NMO equation (A.1) which incorporates a $\frac{1}{v_s^2}$ term. Because the NMO correction is less sensitive to reflections having high velocities than reflections having low velocities, the high end of the velocity spectrum is more widely spaced than the low end.

The height of the time window governs the amount of data needed for computing the coherence. To choose the height of the time window, we must construct the window around a single reflection. The height of the window depends on the frequency content of the reflection data, namely, the lower its frequency content, the greater is the height of the window. The ratio between the time sampling in a common midpoint gather and the time sampling in a semblance image is, in general, between five and ten, namely, the time sampling in the semblance image is five to 10 times bigger than that in the common-midpoint gather.

Appendix B

BACKPROPAGATION ALGORITHM

The procedure that teaches a neural network to map input into desired output patterns requires adjustments of the weight values defining the network. Such adjustments are performed using a gradient-descent method called the backpropagation algorithm.

The output of the network is computed through a forward calculation, namely, going from the input layer to the output layer. Then, the weights of the network are changed using the backpropagation algorithm by reversing the calculations, namely, going backwards from the output layer to the input.

These cycles of forward and backward computations (called **iterations**) are repeated until the actual outputs approach the desired output patterns asymptotically.

B.1 Forward Computations

The forward computations that produce the output o_j for the jth node comprise two steps:

1. Using of the outputs from n other nodes (located in preceding layers), the weighted sum net_j is calculated as input into the jth node,

$$net_j = w_{j0} + \sum_{i=1}^{n} w_{ji} o_i, \tag{B.1}$$

where w_{ji}[1] is the weight from node i (located in one reference layer) to node j (located in any layer forward from the reference layer), o_i is the output of node i and w_{j0} is the bias into node j.

2. net_j is then passed into an **activation function** that generates the output o_j for the jth node:

$$o_j = f(net_j). \tag{B.2}$$

A typical activation function is the **sigmoid function**:

$$f(net_j) = \frac{1}{1 + e^{-net_j}}, \tag{B.3}$$

of which the derivative can be expressed

$$f'(net_j) = f(net_j)(1 - f(net_j)). \tag{B.4}$$

B.2 Backward Computations

In the backpropagation algorithm, an **error function** $E(\vec{w})$ (also called a **cost function**), is minimized in order to bring the actual output as close as possible to the desired output. For any single input pattern, the error function is defined as the least-square difference between the actual and the desired outputs:

$$E(\vec{w}) = \frac{1}{2}\sum_{j=0}^{N}(d_j - o_j(\vec{w}))^2 \tag{B.5}$$

[1]The weights w_{ji} are always defined in the forward direction from the input to the output layers (w_{ij} is never calculated in the reverse direction).

where d_j is the desired output for the jth node of the output layer and N is the number of output nodes. The **error function** E, thus, depends on the parameter vector $\vec{w}$ that is made up of all the weights w_{ji} defining the neural network.

B.2.1 General Expression for the Weight Update

The weight update Δw_{ji}, the incremental change in the weights w_{ji} from one backward computation at iteration $t-1$ to the next backward computation at iteration t, is expressed as follows:

$$w_{ji}(t) = w_{ji}(t-1) + \Delta w_{ji}(t). \tag{B.6}$$

Because a gradient-descent method is used to minimize the error function E in the weight space, the weight update for the connection between node i and node j is expressed as:

$$\Delta w_{ji} = -\alpha \frac{\partial E}{\partial w_{ji}}, \tag{B.7}$$

where α is a coefficient of proportionality. The backpropagation algorithm computes the gradients $\frac{\partial E}{\partial w_{ji}}$ at each computing element and for each weight to that element, starting from the output layer and going backward to the first hidden layer. By the chain rule, equation (B.7) becomes:

$$\Delta w_{ji} = -\alpha \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial net_j} \frac{\partial net_j}{\partial w_{ji}}. \tag{B.8}$$

Equation B.8 is the general expression for the weight update. With w_{ji} defining the connections between node i and node j, the partial derivatives on the right-hand side of equation (B.8), yield different results depending on the location of the jth node. Therefore, we distinguish two cases for calculating the weight update: 1) when the jth node is in the output layer, 2) when the jth node is in a hidden layer. It is the computation of the partial derivative $\frac{\partial E}{\partial o_j}$ in equation (B.8) that differentiates these two cases. The calculations for each case are detailed in the next two sections.

Case 1: Weight Update for the Weights Connected to the Output Nodes Using equations (B.5), (B.4), and (B.1), the three partial derivatives in the right-hand side of equation (B.8) then become:

$$\frac{\partial E}{\partial o_j} = -(d_j - o_j), \tag{B.9}$$

$$\frac{\partial o_j}{\partial net_j} = f'(net_j) = f(net_j)(1 - f(net_j)), \tag{B.10}$$

$$\frac{\partial net_j}{\partial w_{ji}} = o_i. \tag{B.11}$$

Therefore, the weight update Δw_{ij} becomes:

$$\Delta w_{ji} = \alpha(d_j - o_j)f'(net_j)o_i. \tag{B.12}$$

Case 2: Weight Update for the Weights Connected to the Hidden Nodes Once having computed the weight updates starting at the output layer, the the weight updates from the hidden-layer nodes are computed moving backwards through the neural network to the input layer. In computing the weight update to each hidden node, it is necessary to compute the gradients $\frac{\partial E}{\partial w_{ji}}$ for all connections to that node. These backward calculations are performed until the weight updates for all connections are completed. In any hidden node, the partial derivative $\frac{\partial E}{\partial o_j}$ is computed as follows:

$$\frac{\partial E}{\partial o_j} = \sum_{k}^{\kappa} \frac{\partial E}{\partial net_k} \frac{\partial net_k}{\partial o_j}, \tag{B.13}$$

or using equation (B.1) to express $\frac{\partial net_k}{\partial o_j}$:

$$\frac{\partial E}{\partial o_j} = \sum_k^{\kappa} \left(\frac{\partial E}{\partial net_k} \right) w_{kj}, \tag{B.14}$$

where k through κ are the indices of the nodes to which the jth nodes connects. Therefore the weight update from the node i towards the hidden node j is

$$\Delta w_{ji} = -\alpha \sum_k^{\kappa} \left(\frac{\partial E}{\partial net_k} \right) w_{kj} f'(net_j) o_i. \tag{B.15}$$

Weight Update: One Simplified Formula A new quantity called the credit δ_j is introduced, and combines all cases for the weight update into a single equation. With $\delta_j = -\frac{\partial E}{\partial net_j}$, it can be shown that this single equation is:

$$\Delta w_{ji} = \alpha \delta_j o_i, \tag{B.16}$$

where for case 1 involving connections to the output layer, the credit δ_j is expressed:

$$\delta_j = (d_j - o_j) f'(net_j), \tag{B.17}$$

and for case 2 involving connections to each hidden layer, the credit δ_j is

$$\delta_j = \sum_k^{\kappa} \delta_k w_{kj} f'(net_j), \tag{B.18}$$

where k through κ are the indices of nodes to which the jth node connects.

B.2.2 Inertia

Inertia is an extension to the backpropagation algorithm that serves to improve **convergence performance**[2] and to avoid **local minima**[3]. **Inertia** (also called **momentum**) is a term that is added to the weight update in order to smooth the weight changes over time. Therefore, the defining equation for the weight update (B.6) is modified to include this inertia term as follows:

$$w_{ji}(t) = w_{ji}(t-1) + \Delta w_{ji}(t) + \gamma \Delta w_{ji}(t-1), \tag{B.19}$$

where $\gamma \Delta w_{ji}(t-1)$ is the inertia term and γ is the inertia constant in the interval [0,1].

[2]The **convergence performance** is the number of iterations (cycles of forward and backward computations) required during the training phase in order for the actual outputs to approach the ideal outputs adequately for a given training set.

[3]A **local minimum** is a non-optimum solution to the learning problem as opposed to the best solution that is called the **global minimum**.

Appendix C

NEURAL-NETWORK GLOSSARY

Activation functions: A class of functions that is used in the neural-network computations to generate an output signal from each **computing element** by scaling the dot-product signal input to that element. An example of such a function is the **sigmoid fuction**. *See* **Sigmoid function.**

Backpropagation algorithm: A special type of **learning algorithm** used to update the **weights** by minimizing the **l-2 norm** between actual and expected outputs in a multi-layer neural network. This minimization process is performed through a **gradient-descent** optimization technique using a **training set** of data.

Bias: A numerical value describing the **connection** between a "dummy" **node** having a constant value of one and a **computing element.**

Computing elements: Memory processors that comprise the neural network. Each processor receives one or more inputs from other computing elements or from external sources (where the input is called the input vector). The input vector from these sources is then multiplied by its corresponding **weight vector**. The result of this dot-product operation is changed by the **activation function** into the output signal which is either distributed to other processors or becomes the final output of the neural network.

Connection: Is a path between two nodes in the neural network through which the numerical output from one **node** is transmitted to the second node. The connection is described by a numerical scalar value called the **weight**.

Connectivity: Is the linkage structure between **processing elements** or nodes. Nodes may be **fully-connected, locally-connected** to neighboring nodes, or **sparsely-connected** to a few distant nodes.

Convergence or Convergence performance: Number of **iterations** required during the **training phase** for the actual outputs to approach the expected outputs asymptotically using a given **training set**.

Cost function: A function that expresses the mean-squared difference between actual and expected outputs. During the **training phase**, this difference is minimized using the **backpropagation** algorithm.

Degrees of freedom: The number of independent **weights** and **biases** in a neural network.

Epoch: Presentation of all the input **patterns** in a given **training set** to the neural network during the **training phase**.

Error function: *See* **Cost function**.

False alarm: Incorrect output produced by the neural network from an input **pattern** *not* in the **training set**.

Feature: *See* **Local feature**.

Feature detector: Is a spatial filter which when convolved with the input **image** yields an output indicating the presence or the the absence of a particular

feature.

Feature map: The domain that is constructed through the application of a **feature detector** to the input **image**, indicating the approximate locations of a particular **feature.**

Free parameters: *See* **Degrees of freedom.**

Fully-connected *See* **Full connectivity.**

Full connectivity: Is the linkage structure in the neural network in which all **computing elements** or **nodes** are connected to all other computing elements.

Generalize *See* **Generalization.**

Generalization: The ability of a trained system to produce expected outputs using new inputs (that are *not* in the **training set**).

Generalization performance: Is a measure of how well the network system determines the expected output **patterns** from input patterns *not* in the **training set.**

Global minimum: The position on the error surface where the difference between the outputs of the neural network and the desired outputs is the smallest in any place on the surface. The best possible solution for **teaching** the neural network.

Good detection: The expected output produced by the neural network from an input pattern *not* in the **training set.**

Gradient-descent technique: A method of minimizing the **cost function** based on the computation of the derivatives of this function.

Hidden layer: Is a layer made up of **processing elements** inserted between the **input** and **output** layers of a neural network.

Hidden nodes: **Processing elements** that make up the **hidden layer**.

Hierarchical structure: A data processing system organized in layers or levels where data is passed through the system from lower to higher levels. At each succeeding level the data are acted upon and modified by increasingly complex functions. A vision neural network is an example of such a structure where image features at the input are detected and their approximate locations are constructed in a feature map in the first layer. These feature maps (each individual map representing a different feature) are then combined into higher-order feature maps in the second layer.

Image: An ensemble of 2-dimensioned-ordered data elements, or pixels.

Image class: Describes an abstract image representation that is "averaged" out from a suite of images placed as input to a pattern classifier. An image belongs to a class if the output from the pattern classifier approximates the expected output for that class with an error of fit less than some preestablished error threshold.

Image classification: The process by which a **pattern classifier** fits an input image into one of several fixed (pre-established) images classes.

Input layer: Defines the **pattern** presented as input to the network. It is made up of **nodes** which are never computing **elements**.

Inertia: A term added in the **weight-update equation** to smooth the **weight-update** over time.

Iteration: A cycle of forward and backward computations through the neural network.

L2-distance between two vectors: $\sqrt{\sum_{i=1}^{N}(a_i - b_i)^2}$ where $\{a_i\}$ and $\{b_i\}$ represents the components of the two vectors $\vec{a}$ and $\vec{b}$ of dimension N.

L2-norm between two vectors: $\frac{1}{N}\sum_{i=1}^{N}(a_i - b_i)^2$ where $\{a_i\}$ and $\{b_i\}$ represents the components of the two vectors $\vec{a}$ and $\vec{b}$ of dimension N.

Layers: Subsets of **processing elements** or **nodes** in a neural network There are three types of layers:

1. The **input layer**,
2. The **output layer**,
3. The **hidden layers** which are the layers positioned between the input and the output layers.

Of these three layers, only the nodes that make up the input layer are not computing elements.

Learning: *See* **Training**.

Learning algorithm: Is a mathematical process for updating the **weights** in response to input signals applied to the neural network.

Learning criteria: A mathematical condition which, when reached, terminates the learning algorithm and indicates that the neural network has properly learned. These conditions can be either:

1. **l2-norm** reaching a value below a user-defined threshold,
2. Maximum **node-error** reaching a value below a user defined-threshold.

Learning performance: The minimum number of **epochs** required for the neural network to reach the chosen **learning criteria** during the **training phase**.

Local feature: A distinct geometrical shape of limited area inside an input **image** or a hidden feature map.

Locally-connected: *See* **Local connectivitity**.

Local connectivity: Linkage structure in a multilayer neural network in which the **computing elements** in one layer are connected to only a neighborhood subset of nodes in the source layer.

Local minimum: A non-optimum solution in **training** the neural network as opposed to the optimum solution which is called the **global minimum**.

Low and high-order information: Describes the density representation of information from a low density distribution such as that occurring in an image to a higher density distribution such as that illustrated by higher-order feature maps in a vision neural network.

Mapping: Is one-to-one transformation of data from one domain to another.

Mean-square difference: *See* **L2-norm**.

Memorization: The inability of the neural network to give the expected outputs to input **patterns** which are *not* in **the training set**.

Momentum: *See* **Inertia**.

Neural Network: Is an I/O system composed of many **nodes, fully, locally** or **sparsely** connected.

Node: An element in the neural network. The **hidden layers** and the **output layer** are made of nodes which are **computing elements** in contrast to the **input layer** where the nodes have no computing capacity.

Node-error: The absolute difference between an output-node value and its expected value.

Nonweight-sharing connectivity: Is one type of **local connectivity** between **layers** in a neural-network in which the **weight filter** vector of each **node** in a destination layer is changed as each **receptive field** in the source layer is passed through a **scanning** process.

Nontraining image samples: *See* **Untrained image samples.**

Output layer: Is the last layer in the neural network in which the values of the **nodes** represent the final output **pattern** of the network.

Partial connectivity: *See* **Local connectivity.**

Pattern: Ensemble of input or output node values of a neural network forming a distinct shape.

Pattern classifier: A neural network which processes an input **pattern** and places it into one of several classes.

Pattern size: Number of nodes in a **pattern**.

Pattern transformer: Transforms input **patterns** to output patterns having the same pattern size.

Processing element: *See* **Computing elements**

Receptive field: Is a subset of neighboring nodes in the source layer. This subset is used to produce a single node in the destination layer through application of a weight filter.

Reinforcement learning: A type of **supervised learning** in which feedback indicates whether or not each output is correct. Another name for this type of learning is *learning with a critic.*

Root-mean-square difference *See* **L2 distance.**

Scanning process: A procedure in which a window moving through the source layer generates nodes in the destination layer through the application of one or several weight filters.

Sigmoid function: An activation function expressed as $f(x) = \frac{1}{1+e^{-x}}$.

Sparsely connected: *See* **Sparse connectivity.**

Sparse connectivity: The linkage structure in the neural network in which the **processing elements** or nodes in the neural network are connected to only a few distant processing elements.

Squashing function: *See* **Activation functions** and **Sigmoid function.**

Supervised learnable system: Is a system which is trained according to a **supervised learning** paradigm.

Supervised learning: A type of **learning** in which the actual output of the network is compared with an expected output, and the **weights** in the neural network are adjusted accordingly.

Teaching paradigm: *see* **Training paradigm.**

Testing phase: Procedure for verifying whether a trained network gives the expected outputs from input **patterns** *not* in the **training set**.

Time-sequence predictor: Is a special type of neural network in which the input is a time-series defined from some past moment in time to the present time. The output from this network will represent a time series which is an extrapolation of the input to some future time.

Training: Is the adjusting of the network **weight** and **bias** values until the selected **learning criteria** indicates that the network has learned properly. The network is trained using a set of sample inputs patterns called the **training set**.

Training paradigm: A mathematical model used to teach the neural network to process an **input pattern** and yield a given output. There are three classes of training paradigms:

1. Supervised training,
2. Unsupervised training,
3. Reinforcement training.

Training phase: The procedure for **training** the neural network using a **training set**.

Training samples: *See* **Training set**.

Training set: A set of input/"desired" output patterns used during the **training phase** to adjust the **weights** and **biases** until the desired output pattern is achieved.

Unsupervised learning: A method of **training** neural networks which requires only input patterns but *not* expected output patterns.

Untrained input samples: Sample image **patterns** which are not in the **training set**.

Weights Are numerical values (also called weight values) describing the connections between **nodes**. The numerical information passed from one node to another and multiplied by the weight value describes the **connection**.

Weight filter: Set of weights associated with a **computing element** describing the connections between the given element and some neighboring **nodes** in the source **layer**.

Weight-sharing connectivity: Is one type of **local connectivity** between **layers** in a neural-network in which the **weight-filter** vector of each node in a destination layer is constrained to remain the same as each **receptive field** in the source layer is scanned.

Weight vector: Can represent either all the **weights** defining the neural network or all the weights associated with a **computing element**.

Weight-update: The incremental change of the **weight** value of each connection from one iteration to the next during the **training phase**.

Weight-update equation: Is the equation $w_{ji}(t) = w_{ji}(t-1) + \Delta w_{ji}(t)$, governing the change of weights in the neural network from one **iteration** to the next during the **training phase**.

Appendix D

SEISMIC GLOSSARY (AFTER SHERIFF 1978)

A.V.O.: Amplitude Versus Offset: The study of how the amplitude of seismic reflection events varies with distance from the source.

Coherence: *See* **Semblance**

Constant velocity stack: A **stack section moved-out** with only one constant velocity function.

Common-midpoint or CMP gather: The set of traces which have a *common midpoint.*

Common-midpoint: Having the same midpoint between source and detector.

Common-midpoint stack: A sum of traces corresponding to the same midpoint. The traces from different shot profiles having different offset distances are gathered together, corrected for normal-moveout and then summed (stacked). The objective is to attenuate random effects and events whose dependence on offset is different from that of primary reflections.

Constant velocity stack: A **stack section moved-out** with only one constant velocity function.

Diffraction: An optical phenomenon that cannot be explained by geometric optics, but requires wave theory for its correct description. For example, the bending of the wave energy around obstacles does not follow Snell's law, but is explained completely by Huygens principle.

Dipping event: Reflection coming for a reflector which is dipping.

Dip-moveout: A change in the arrival time of a reflection because of dip of the reflector.

First break picking: Consists in selecting the time of the first recorded signal attribuable to a seismic-wave travel from a known source. First breaks on reflection records are used for information about weathering.

Geophone: The instrument used to transform seismic energy into electrical voltage.

Multiple: Seismic energy which has been reflected more than once.

Normal moveout: NMO. The variation of reflection arrival time because of shotpoint-to-geophone distance (**offset**).

Normal moveout correction: The time correction applied to reflection time because of normal moveout.

Primary event: Reflection coming from a reflector without multiples.

Reflection: The energy or wave from a seismic source which has been reflected (returned) from an acoustic-impedance contrast (**reflector**) or series of contrasts within the Earth.

Seismic wave: An elastic disturbance which is propagated from point to point through the medium.

Seismogram: A plot of seismic traces from a single point source.

Semblance: A measure of multichannel coherence; the energy of a sum trace divided by the mean energy of the components of the sum. It is basically the energy of the stack normalized by the mean energy of the components of the stack.

Shot record: The recording of seismic data from one shot (source) picked up by a spread of geophones.

Sloth: Reciprocal of squared velocity.

Stack: A composite record made by combining traces from different records.

Stacking process: The process that creates a **stack**.

Stacked section: *See* **Stack**

Stacking velocity: Velocity calculated from normal-moveout measurements and a constant-velocity model. It is used to maximize events in common-midpoint stacking.

Trace editing: Consists in deleting unwanted seismic traces having a low signal/noise ratio.

Velocity picking: Consists in selecting the time-velocity pairs for **N.M.O.**.