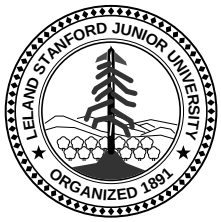


STANFORD UNIVERSITY

ADAPTIVE NEURAL NETWORKS FOR MINE DETECTION

Bernard Widrow, Principal Investigator (415-723-4949)

Takeshi Doi
Gregory L. Plett
Information Systems Laboratory
Stanford University
Durand Bldg. Rm 104
Stanford, CA 94305-4055
(415) 723-4769

April 28, 1995

Scientific and Technical Report
Final Report
Contract DAAK70-92-K-0003 (03/27/92 - 03/31/95)

UNCLASSIFIED

Prepared for
U.S. Army Belvoir Research, Development, and Engineering Center
Countermine Systems Directorate
Fort Belvoir, VA 22060

Contents

1 Introduction

The detection and disposal of anti-personnel land mines is a problem of significant military, economic and humanitarian concern. Not only do they pose a deterrent to military activity during conflict, they also remain lethal long after that conflict is over. The statistics are staggering; [7] eighty percent of the victims of anti-personnel mines are civilians, many of whom are children. The number of such mines currently in place is estimated at over 100 million [8], and is increasing at the rate of one half to one million mines per year.

There is great motivation to research new countermine systems. Current anti-personnel land mines cost between \$3–\$25 per mine and clearance methods cost between \$300–\$1000 per mine removed. The rate of removal is too slow, and the percentage of mines cleared is too low to be acceptable. A mine detection system which will operate with a high degree of accuracy while being simple enough to deploy at low cost is plainly needed. Successful detection methods could result in a considerable reductions of battlefield and civilian casualties, and less costly minesweeping operations.

One of the problems faced by the countermine system is that newer mines contain very little metal, and thus it is not feasible to use conventional metal detectors to locate them. A more sophisticated approach is taken in this work, where we train adaptive neural networks (NNs) to detect land mines using data collected from a mine lane facility located at Ft. Belvoir. This data consists of a set of measurements made with a Separated Aperture Sensor operating in the "waveguide near cutoff" mode. Previous work has shown that neural network pattern classifiers can flawlessly detect anti-tank mines [30], and the present work addresses the much harder problem of detecting smaller anti-personnel mines.

A complete discussion of the theory of neural networks and their uses in pattern recognition is beyond the scope of this document. For further information, the reader is referred to [43, 22, 48, 49]. For our purposes, it is sufficient to view a neural network as a device to perform a functional mapping between an input vector X and an output vector: $Y = \text{fn}(X)$. For our application, we would like to present patterns X to the network which correspond to measurements over mines and have the output $Y = \text{True}$. Likewise we would like to input patterns X corresponding to measurements not over mines (over background) and have the output $Y = \text{False}$.

Since neural networks are trainable universal function approximators, they can be taught using empirical data to learn a decision rule and accurately classify input vectors into various categories. Minimal human knowledge of the dynamics of the underlying phenomenon is needed for this task. This makes neural network pattern recognition particularly desirable in situations where the measurement vector X is large, or when the structure of the probability distributions for each class is complex or unknown. In either of these cases it is difficult and perhaps impossible to come up with an analytical solution.

This report documents the work done at Stanford University under contract DAAK70-92-K-0003 over the period 3/27/92 to 03/31/95. The scope of the work performed is quite

large, and an effort has been made to describe it in a methodical and logical manner. Therefore, subjects that relate to each other are described together even if the work was performed months apart. For a more chronological presentation of the work, the reader is referred to [11, 12, 14, 13, 15, 16, 18, 17, 19, 20, 21]. In this text, the Ft. Belvoir mine lane facility is discussed first, followed by the methods used to measure the raw data. Next, neural networks and probabilistic neural networks are discussed, along with the training methods used. Preprocessing and postprocessing methods are described, the outlier removal and transformation preprocessing techniques are elaborated on. Results are discussed, and in conclusion the proposed network architecture is presented.

1.1 The Ft. Belvoir Mine Lane Facility

This work is based on measurements made at a simulated mine lane facility located at Ft. Belvoir. The mine lane is housed in a quonset hut whose floor is divided into a number of strips of different soil types. In these lanes, different sized and shaped objects made from various materials (which represent actual mines in terms of their dielectric qualities) are buried. Over the course of this contract, four sets of measurements, titled "SAS3"–"SAS6", were made at this facility. These data sets were measured for a single mine lane, but with different sensor configurations[1]. After initial testing, work on this project focused exclusively on using the last set of measurements, "SAS6", which was shown at an early date to give the best results.

The "SAS6" data was measured from a 60'×4' mine lane of sandy soil. Five different types of simulated mines (targets) were buried:

- Type 1: 6" diameter × 3" wood disk.
- Type 2: 5" diameter × 3" hard plastic disk.
- Type 3: 3.5" diameter × 3" nylon disk.
- Type 4: 6" × 6" × 3" wood block.
- Type 5: 4" × 2" "butterfly" shaped target.

There were a total of 114 targets: 15, 30, 12, 15 and 42 of each mine type, respectively. Appendix A lists the location in the mine lane of each of the targets.

1.2 Measurement Methods

An experimental apparatus at Ft. Belvoir was used to obtain the training and test data. The theory and data collection apparatus is described in [4, Section 2] for the "SAS1" and

[1] The data sets "SAS1" and "SAS2" were measured for a different mine lane (for contract DAAK70-89-K-0001[30]), where the lane contained simulated anti-tank mines. This contract focuses entirely on the more difficult problem of locating anti-personnel mines.

"SAS2" measurements. Some changes were made in the measurement apparatus for the most recent sets of data, but the theory remains the same. The sensor system used is also described in [39]. Briefly, the sensor and measurement apparatus works as follows:

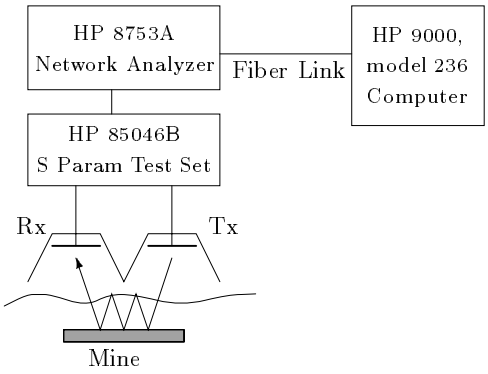


Figure 1: Measurement Setup

A block diagram of the measurement system is shown in Figure 1. A Hewlett Packard 8753A Network Analyzer is the heart of the system. It is capable of generating sinusoidal signals between 300 kHz and 3 GHz. It is attached, through an HP 85046B S parameter test set, to a separated aperture sensor. The sensor, located (nominally) 2" above the ground, consists of a transmitting aperture or horn which injects microwaves into the soil, and a receiving aperture which provides a return signal to the Hewlett Packard equipment. For the "SAS6" data used during this period, the resonant frequency of the horns was 1 GHz and the septum width (the width between transmitting and receiving horns) was 0". In previously measured data sets the septum width and resonant frequencies were different. For example, the septum width was 6" for the "SAS5" measurements. Figure 2 shows a pictorial diagram of the sensor. The horns look like metal troughs, and the dipole antennae run longitudinally.

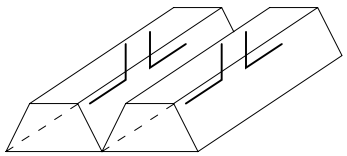


Figure 2: Microwave Horn Antennae

Both the signal received by the "receiving" horn and the return signal picked up by the transmitting aperture are sensed. The equipment records the complex-valued two-port scattering parameters or S parameters (roughly analogous to the Z parameters of a low-frequency circuit) of the sensor.

The tests use continuous wave sine waves and standing waves were detected and used

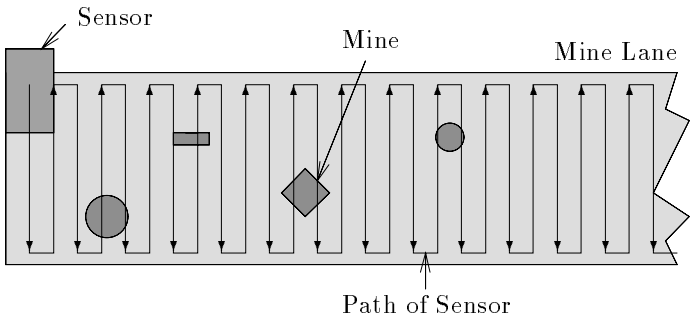


Figure 3: Measurement Method

to compute the S parameters. The measurements were repeated five times and averaged to reduce noise. In "SAS1" and "SAS2", the data was averaged 15 times. There was no noticeable difference between averaging 5 and 15 times, which is not surprising since some implicit averaging takes place when detecting standing waves. The horns were connected directly to the S parameter test set.

Measurements were made on a grid of 1.5" $\times$ 1.5". There were 27 columns and 527 rows of measurements. At each point, all S parameters were measured (S_{11},S_{12},S_{21},S_{22}). Eleven different frequencies in 40MHz intervals were used ranging from 800MHz to 1200MHz, inclusive. At each point, the magnitude and phase component of each S parameter was measured. So, there were a total of 88 ($4\times11\times2$) measurements at each location in the grid. Figure 3 shows how the sensor head was scanned over the mine lane.

Unlike the system used to measure the "SAS1" and "SAS2" data, which used a wheeled cart to house the data collection equipment, the more recent data was collected by a system hung on an overhead rail. The measuring system was controlled at a distance by an HP 9000 computer via optical fiber (HP-IB) link. The computer was able to move the sensor head laterally with a worm gear control and record all measurements automatically. To move the sensor longitudinally (from row to row), a human operator had to move the apparatus the required 1.5" along the overhead rail.

1.3 Measurement of S Parameters

The measurements conducted at Ft. Belvoir used the HP S parameter test set to automatically measure the S parameters of the two port network formed by the microwave horns and the earth waveguide. When a system is built for the field, however, it will be necessary to build hardware to measure these parameters. This section describes one way it can be done.

If the output[2] of the two-port network is terminated in its matched impedance Z_0

[2]The two-port network we are using is symmetric, so the concept of "input" or "output" only refers to the way the network is configured for measurement. Typically the "input" to the network, which we sometimes

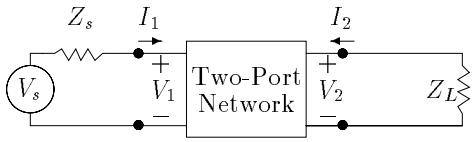


Figure 4: Measuring S Parameters

(such that nothing is reflected from the load), then the S parameters have the following interpretation[3]:

S_{11} = Input reflection coefficient with the output port terminated in Z_0.
S_{21} = Forward transmission coefficient with Z_0 load.
S_{22} = Output reflection coefficient with Z_0 source and $V_s = 0$.
S_{12} = Reverse transmission coefficient with the source impedance = Z_0.

Without knowing the matched impedance Z_0 of the output, it is not possible to measure the S parameters directly. Therefore, we must use an indirect method which works as follows: First, an intermediate set of parameters must be measured. These are the "h parameters" of the network. Given a two-port network as shown in Figure 4 which is driven by a sinusoidal source at a fixed frequency, the h parameters are:

$$\begin{aligned} V_1 &= h_{11}I_1 + h_{12}V_2 \\ I_2 &= h_{21}I_1 + h_{22}V_2 \end{aligned}$$

To measure the h parameters, a two step process is performed. First, the output of the two-port network is shorted. $V_2 = 0$ and h_{11} and h_{21} can be obtained:

$$\begin{aligned} h_{11} &= V_1/I_1 \\ h_{21} &= I_2/I_1 \end{aligned}$$

Likewise, if we open circuit the input making $I_1 = 0$ and drive the output terminal, we get:

$$\begin{aligned} h_{12} &= V_1/V_2 \\ h_{22} &= I_2/V_2 \end{aligned}$$

We can now obtain the S parameters from the h parameters:

$$\begin{aligned} S_{11} &= \frac{(h'_{11} - 1)(1 + h'_{22}) - h_{12}h_{21}}{(1 + h'_{11})(1 + h'_{22}) - h_{12}h_{21}} \\ S_{12} &= \frac{2h_{12}}{(1 + h'_{11})(1 + h'_{22}) - h_{12}h_{21}} \end{aligned}$$

call the transmit head or port 1, is driven by a sinusoidal source, and the "output", which we sometimes call the receive head or port 2, is used to determine the response of the network to that source.

[3]If the output is not terminated in Z_0 then the parameters are still meaningful, but their interpretation is not apparent.

$$S_{21} = \frac{-2h_{21}}{(1+h'_{11})(1+h'_{22}) - h_{12}h_{21}}$$
$$S_{22} = \frac{(h'_{11}+1)(1-h'_{22}) + h_{12}h_{21}}{(1+h'_{11})(1+h'_{22}) - h_{12}h_{21}}$$

Note: $h'_{11} = h_{11}/R_0$; $h'_{22} = h_{22}R_0$, where R_0 is the resistance of the source, and is a quantity known for any given oscillator.

The following section describes how to measure a complex voltage or current as required for measuring the h parameters[4]. It may seem odd, though, that actual current is flowing, since no real closed circuit seems to exist (i.e. the oscillator is connected to wires which dangle in the air). However, we must first realize that the familiar concepts of circuits are simplified approximations to Maxwel's equations specifically modified for low frequency operation, with minimal power loss due to radiation. The purpose of our antenna system is to radiate energy, so we must discard those concepts. Real (AC) current does indeed flow in the system, and charge is alternately deposited and removed from the ends of the dipole antenna. There are real voltage differences across the antenna, and both the current and voltage can be measured in standard ways using standard equipment.

The energy radiated from a dipole antenna is transmitted in both electric and magnetic waves. One researcher found that the leakage mode which propagates between the ground surface and the septum was a TE mode wave [9, pp. 2-5]; the detection mode, however, was largely transmitted through the soil over the target. There was some energy inside the target, with the strongest field component parallel to the dipoles (again, a TE mode).

1.3.1 Design of a Complex Voltmeter and Ammeter

One of the requirements for measuring the h parameters is a voltmeter which will measure "complex" voltages. In actuality, all voltages are real, and the concept of a complex voltage is just a mathematical convenience which incorporates both the amplitude and phase of a sinusoidal signal (where the phase is measured relative to the source).

Such "vector" voltmeters do exist as off-the-shelf items, but it turns out that their design is simple enough that building one which would even provide digital output for computer I/O would be no great difficulty. The design presented here may be a little crude (i.e. a microwave designer may find some places where special measures must be taken), but with today's advanced op-amps and analog circuitry which work at very high operating frequencies, it will probably be sufficient.

First, let us concern ourselves with measuring the amplitude of the sinusoidal signal. The circuitry to perform this is shown in the top section of Figure 5. The first stage of the process is to buffer and amplify the input signal so that no appreciable load is added to

[4]Note that we need to measure complex values for the h parameters even if we only use the magnitude of the S parameters since the magnitude of the S parameters depends on both the magnitude and phase of the h parameters.

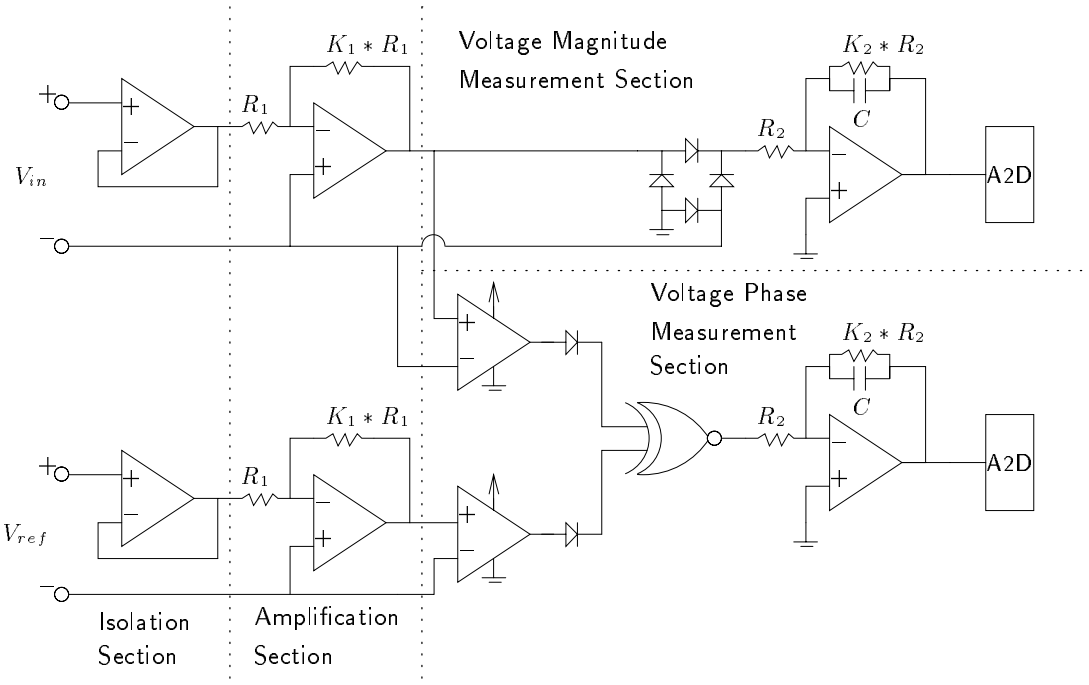


Figure 5: Circuit Diagram for Complex Voltmeter

the system we are measuring, and so that we amplify the signal to a value which will be within the range of our measurement system. The first op-amp is configured as a voltage follower whose input resistance is nearly infinite and whose output resistance is near zero. The second op-amp amplifies the input by $-K_1$. The factor of '-1' does not affect our analysis since the input sinusoid is symmetric around zero volts. However, the amplitude scaling factor of K_1 must be taken into account when interpreting the final output of the system.

We would like to determine the magnitude A of the sinusoidal signal. To do this, the amplified signal is passed through a full-wave rectifier. The output of the full wave rectifier, which we will call $X(t)$, is shown in Figure 6. $X(t)$ is periodic with period $1/(2f_o)$ if f_o is the frequency of the input signal, and with amplitude $A = K_1|V_{in}|$. The average value of $X(t)$ can be found to be $2A/\pi$. We can measure this average value and thus compute A as follows: since $X(t)$ is periodic, it will have a Fourier Series, and the DC term will be the average value. So, to determine A, we must low-pass filter $X(t)$ to extract the DC component. A question of practical interest is "How narrow a filter do we need?" It turns out that the filter is very realizable. Since the input is periodic, frequency components will occur at multiples of the input frequency and nowhere else. Since the input frequency is about 1GHz, we just need a low pass filter with a cutoff somewhere below 1GHz. This is certainly trivial to build! We could use a simple single stage filter with cutoff frequency as

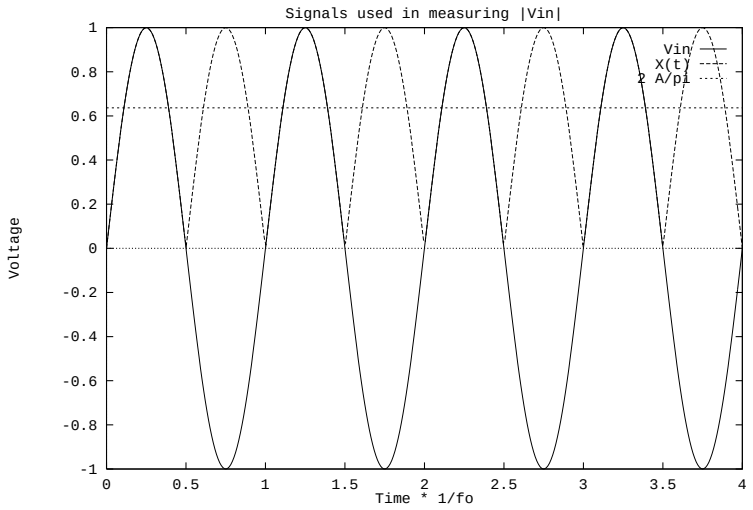


Figure 6: Signals used to determine voltage amplitude.

high as 1kHz, and have a suppression of more than 120dB on all other components[5]. The rightmost op-amp in Figure 5 is configured as a low-pass filter with gain K_2 and its cutoff frequency is $1/(2\pi R_2 C_2 K_2)$.

So, $X(t)$ is filtered and scaled by a constant K_2 to be in the correct range for an A2D converter which is sampled by the monitoring computer. The sample value is interpreted as being scaled by K_1, K_2 and π. i.e. $|V_{in}| = A2D\frac{\pi}{2K_1K_2}$.

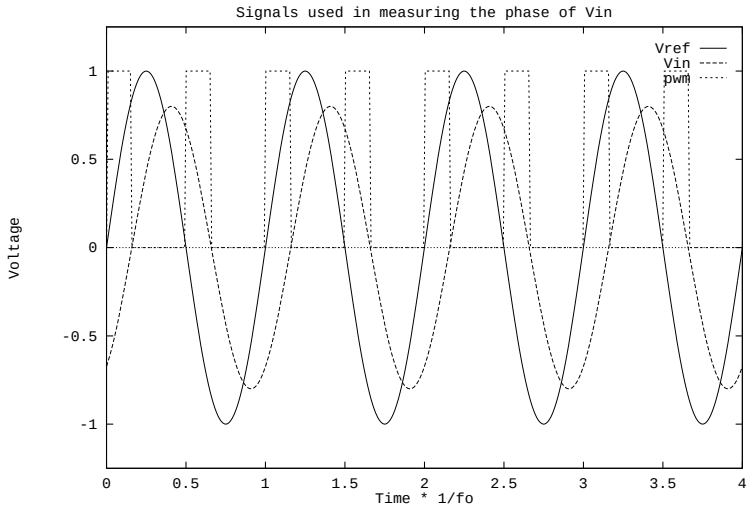


Figure 7: Signals used to determine voltage phase.

[5]Smaller bandwidths are preferred to suppress spurious noise and transient effects. However the settling time of the LP filter is inversely proportional to the bandwidth. Thus, if measurements must be made quickly, then a higher bandwidth may be required.

Now, let us consider how one would measure the phase change with respect to the reference signal. This turns out to be not much more difficult and is shown in the lower half of Figure 5. As before, we buffer the input signal, and we also buffer the reference signal. Both signals are then passed through comparators which realize the function: $y(t) = K_3 \mathrm{sgn}(x(t))$. Next, the output from the comparators are passed through diodes which convert the levels to be compatible with regular logic chips. The two signals are input to an XOR gate[6]. The output of the XOR gate is 1 when the input signals differ, and 0 when they are the same. Thus, if the signals are perfectly in phase, the output of the XOR gate will be identically zero. If they are 180° out of phase, the output of the XOR gate will be identically 1. Otherwise, the output of the XOR gate will be a pulse width modulated signal whose width represents the fraction of 180 degrees the signals are out of phase[7]. An example is shown in Figure 7 where there is a phase shift between the reference and input voltages, and the pulse width modulated output is shown as "pwm". As with the absolute voltage measurement circuit, the average value of this signal is where the meaningful information is contained. Low pass filter the pwm signal with an identical LP filter, and measure the average value with an A2D converter. The output of the A2D converter is a function of the phase deviation, i.e. $phase = A2D \frac{180°}{\text{max A2D}}$.

The preceding analysis has assumed that a steady state sinusoidal signal is being measured. Since there will probably be noise and fluctuations in the signal, some sort of averaging mechanism is desired. But, this is already built into the system! The LP filters will average out a substantial amount of signal fluctuation.

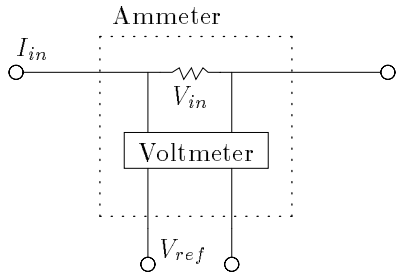


Figure 8: Circuit Diagram for Complex Ammeter

An ammeter can be constructed in exactly the same way as a voltmeter. The very minor difference is that a resistor R_ϵ is introduced into the circuit. This resistor is of diminishingly small value so that it does not disturb the operation of the main circuit. The voltage across this resistor is measured, and the current is calculated as $I = V/R_\epsilon$. To

[6] An XOR gate is used here because of the elegance of the solution. However, getting an XOR gate to work at 1GHz may be tricky. An equally simple solution exists using a logical 'NAND' function which can be constructed from 3 discrete transistors. This can easily be made to work at 1GHz.

[7] Note: in steady state, phase differences of 0 to 180 degrees are indistinguishable from phase differences of 0 to −180 degrees, both mathematically and physically.

measure the phase difference in the current, the voltage across the resistor, V_{in}, is compared with the reference voltage. This is illustrated in Figure 8.

1.4 General Approach to the Problem

This section introduces the general approach to solving the problem of detecting mines with neural networks. The data files used by the neural network simulator and related programs are discussed and a data flow analysis is presented so that the reader can see how preprocessing, postprocessing and training fit into the system. In addition, the test philosophy, desired response calculation and method of reporting results are discussed.

1.4.1 Data Files

The following data files are used:

Raw Data:

The S parameter data measured by personnel at Ft. Belvoir are stored in raw data files, in the format in which it was received. There are 88 files: one file for each measurement type. The magnitude of each S parameter is stored as a dB value, and the phase is stored in radians (between 0 and 2π). In each file, there are 527 rows of measurements, with 27 columns per row.

Globally Preprocessed Data:

While it is possible to use the raw data directly with the neural network, much more favorable results can be achieved by first performing some preprocessing operations on the data. The available options are discussed in Section 4.1. The preprocessed data are stored in 44 files: each contains 527 rows and 27 columns of magnitude and phase information.

Input Vectors:

Multiple S parameters measured at multiple frequencies, all measured at a single spatial location can be combined into a single vector of data. These vectors are, in practice, assembled at run time from the globally preprocessed data files. However, it is convenient for the discussion here to consider them as a single "virtual" file which contains 527 rows and 27 columns of vector data.

KL Transformation Matrices:

In order to use the Karhunen-Loeve transform or any of the other transform methods, one has to first compute the transformation matrices from the input vectors. Being a computationally intensive operation, and one that only has to be done once, this is done off-line, and the results are saved in files.

VQ Centroids:

In order to use the Vector Quantizer with the network, one must compute the centroids

of the input vectors. This is also a computationally intensive operation that only needs to be done once, and it is done off-line, with the results saved to files.

Output:

The simulator output is in ASCII, human readable format. It contains periodic summaries of the simulator state and detection rates.

Miscellaneous Files:

Several other files are used to control the simulations. Among the options which are selectable via these files are: the network architecture to be simulated; the learning method to be used; the frequency/S parameter files to be used as input to the simulation; and lists of preprocessing and postprocessing directives. Another file tabulates all targets and their locations. Its contents are listed in Appendix A.

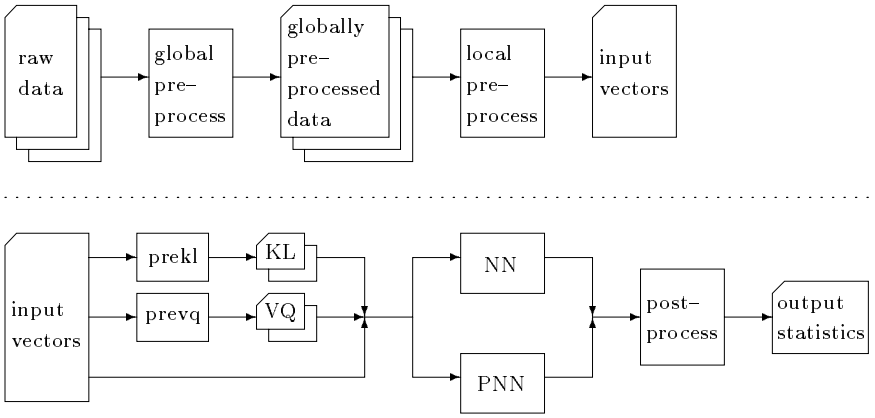


Figure 9: Data flow in the simulation system

In Figure 9, we see how the data files interact with the simulator system. The global preprocessor takes the raw data files provided by Ft. Belvoir, performs global preprocessing operations on them, and writes the resulting data to the globally preprocessed data files. The local preprocessor assembles input vectors and performs local preprocessing on the globally preprocessed data. These input vectors are in turn used by the neural network simulator, the probabilistic neural network simulator, and by prevq and prekl which train the vector quantizers and transform methods respectively. The neural network and probabilistic neural network simulators can also use the transform matrices generated by the prekl program, and the centroids generated by the prevq program. The output of the simulators may be postprocessed, and mine detection statistics are written to the output file. More elaborate descriptions of the individual parts of the simulation system are given in later sections.

1.4.2 Training and Test Philosophy

For the purposes of training and testing, the length of the mine lane is divided up into five sections which are approximately equal in length. The first, middle and last sections are used for training the neural network, and the remaining two sections are used only for testing. The exact divisions, chosen such that no mine is bisected by a region boundary, can be seen in Appendix A. The training and test data are taken from intermingled areas of the mine lane to provide a heterogeneous mixture of terrain and mine types for both the training and test region.

During training, a vector of data taken randomly from the training region of the mine lane is presented to the neural network, and the network output is trained to a positive or negative response depending on whether a mine is present. This pattern of training with random data vectors is repeated until the entire data set is presented to the neural network approximately 100 times.

At predetermined intervals, the weights in the neural network are held constant, and the network is tested on all the possible vectors of data from both the training and test regions in the mine lane. Both the training and test regions are used to evaluate the performance of the neural network to detect mines, but the results from the two regions are reported separately. Thus, the neural network is tested on both the data used for training and data which has never been seen. This crossvalidation approach is a good method of testing the network's ability to generalize its pattern recognition to data which has not been used for training.

1.4.3 Desired Response Calculation

The method used to calculate the desired response for a single spatial point in the mine lane utilized an exact calculation of the mine positions. Since the centers, shape and rotation of each mine are known, the arrangement of the mines can be accurately determined. When training the neural network, each data point corresponding to a 1.5"×1.5" grid square was given a positive desired response if part of that grid square overlapped a mine, and a negative desired response otherwise.

The desired response used when testing the network was somewhat different from the desired response used when training. The concept of an "extended mine region" immediately surrounding a mine was developed. Since a positive response just 1.5" or even 3" away from a mine should not be considered a false positive, the mines could be selectively "enlarged" during testing. If an extended mine region of zero was used, then the desired response for testing was the same as for training. If an extended mine region of 1 was used, then the mine was enlarged by one grid square in each direction by a morphological dilation operation. (by convolution with a 3×3 filter with all values set to 1). If an extended mine region of 2 was used, then the mine was enlarged by two grid squares in each direction by dilating twice. This is shown in Figure 10. The circle represents a land mine and the darkest region

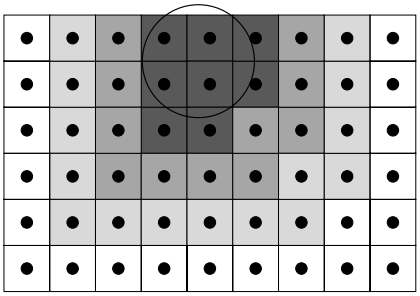


Figure 10: Extended mine regions of size 1 and 2.

consists of the mine data points. The middle shaded region is the extended mine region of size 1, and the lightest shaded region is comprised of the additional data points included for an extended mine region of size 2. Appendix A shows the entire mine lane with all mines and their extended mine regions depicted.

1.4.4 Method of Reporting Results

For every fixed (user specified) number of training iterations[8], a test iteration is performed which tests the network's ability to locate mines. All possible input vectors are presented to the network, and the output is computed. For each positive result, there are two possibilities: either this positive result has in fact found a mine, (a "True Positive") or it is a false alarm (a "False Positive"). In addition, the positive result can occur in either the training or test section of the mine lane. The number of true and false positives in both the training and test sections of the mine lane are written separately to the output file at the end of the test iteration.

The test iteration where the greatest number of true positives occurs in the test section is considered to be the optimum point in the simulation. The reason that the number of true positives in the test section is used as the metric as opposed to those in the training section or, for that matter, the total number of true positives overall, is that any sufficiently large network can completely learn the training data and still give very poor generalization. In a real situation, we will never see inputs exactly like those in the training section, so it is the generalization ability we care most about.

Several different methods of interpreting the results have been used over the course of the contract period. First, we considered the percentage of points that the network correctly classified. However, since we don't need to correctly classify *all* of the points over a mine in order to find the mine, this metric did not give us an accurate feel for exactly how many of the mines were being found. So, the next step taken was to postprocess the responses at each point in the network, looking for groups of positive responses. A contiguous group of 3 or more positive *point* responses was considered a positive *group* response. If the center

[8]Typically a decimation factor of 1000 or 2000 is used for stochastic backprop learning. A factor of 1 is used for batch mode learning.

of this group was within 3 grid points of a mine's center, then this positive *group* response was considered a true positive, else it was considered a false positive. Then, we calculated:

$$TP\% = \frac{\#\text{Mines Found}}{\text{Total Number of Mines}} \times 100$$

$$FP\% = \frac{\#\text{False Alarms}}{\text{Total \# Positive Group Responsess}} \times 100$$

This method worked well, but involved postprocessing the network responses. This may not be practical in a field implementation, so an alternate method was finally adopted. The new method of reporting results involves no postprocessing, and since there is no grouping, the network response at each point is considered by itself. Thus, a mine is "found" if one or more points covered by the (possibly extended) mine is a positive response, and a false positive is any other positive point outside the mine and outside the extended mine region surrounding a mine. i.e.:

1. For every network response that is positive, if it is a point over a mine, or over the extended mine region (of selectable width) surrounding a mine, then mark that mine as found. If the positive response is over background, label that measurement as a false positive.

2. Pick the iteration from the computer run where the generalization peaked. If there are two equivalent points, choose the one which had the smallest false positive rate. If the points are still equivalent, pick the one with the highest training rate, with the smallest number of false positives.

3. Calculate the true positive rate as:

$$TP\% = \frac{\#\text{Mines Found}}{\text{Total Number of Mines}} \times 100$$

Calculate the false positive rate as:

$$FP\% = \frac{\#\text{FP Points}}{\#\text{Background Points}} \times 100$$

Because the method of reporting results has changed, comparing results between different periods is difficult, and only qualitative remarks can be made. Therefore, we only include in this report those results which were generated for the final performance metric. The reader is referred to the other reports [13, 17] for the other results.

2 Neural Networks

Computation by means of networks of artificial "neural" elements is an idea that has attracted a large following among scientists and engineers. Neural networks have proven themselves capable of solving problems that would be impossible or impractical to solve by other means. Many researchers have made observations with respect to a wide variety of neural models, adaptation algorithms, and network architectures. A consensus emerges from this work. The following are valuable attributes of neural networks which are not generally shared by other computing systems:

- Neural networks are not programmed. Rather, they are trained to respond and perform highly refined tasks for which no precise rules exist. One such task is pattern recognition (such as the discrimination of mines in a minefield), in situations where no empirical or theoretical rules are known.

- Neural networks can generalize. That is, they can deliver accurate responses to inputs that were not presented during training. This is a crucially important feature since no system can be trained on all possible mine and background patterns.

- Neural networks can give very high speed response to input stimuli, once trained, due to their inherently parallel nature.

- Neural networks can be continually trained, keeping up with changing input environments.

- Since experience is stored distributively in the connection weights, neural networks are fault tolerant. They are able to adapt around their own internal defects. Graceful degradation results if the network's electronic components begin to fail.

In most applications, neural networks are applied in combination with more conventional computing systems. Neural networks provide the required learning, associative memory, and decision making functions. Conventional computing systems are needed to couple them to existing electronic sensors (to control the measurement devices, and to preprocess the data) and human interfaces (to postprocess the network output and to display where mines have been located); and to act as teachers when the network is undergoing supervised learning.

In the study described in this report, two distinct types of neural networks were applied to the mine detection problem. The multi-layered feedforward neural networks (with all neurons implementing sigmoidal units) and the learning algorithms used are described in this section. The probabalistic neural network which is based on Bayesian classification is described in Section 3.

2.1 NN Structure

Neural networks are interconnected structures of simple processing elements which crudely model the function of a biological neuron. Each artificial neuron (hereafter referred to simply as neuron) has the composition shown in Figure 11. Internally, the scalar product of the input vector[9] and a weight vector is computed, and the output is a non-linear function of this scalar product: $out = F(X^T \cdot W)$. In our work, $F(\cdot) = \tanh(\cdot)$. By modifying the value of the weight vector, different output functions can be realized.

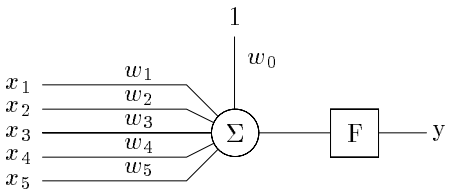


Figure 11: Structure of a Neuron

By combining many of these simple neurons in a layered network, where one element of the input vector of each neuron in a layer is connected to the output of all neurons on the previous layer, a very powerful computational tool is achieved. This structure is shown in Figure 12. It has been shown by Kolgomorov that such a network with a single hidden layer and a sufficient number of neurons is capable of computing (with some set of weight vectors) any continuous nonlinear function to any degree of accuracy. Thus, for example, it is able to compute a discriminant function used for pattern recognition.

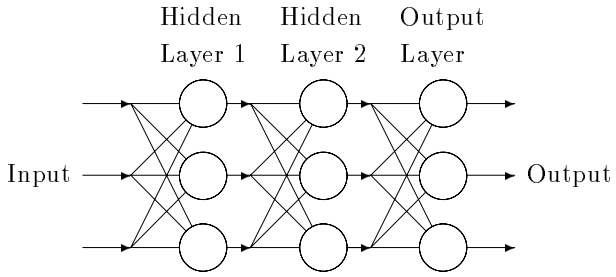


Figure 12: Network Structure

2.2 Weight Initialization

The weights of the NN can be initialized using the weights saved from a previous simulation, or by random initialization. If the random method is used, the weights of each layer are

[9]The input vector is augmented by adding a zeroth element, always equal to 1.

initialized with uniform distribution and zero mean,

$$w \sim \text{unif}\left[+\sqrt{\frac{3}{N}}a, -\sqrt{\frac{3}{N}}a\right]$$

where N is the number of inputs to a particular layer and a is a constant between 1 and 1.5. This provides a variance of a^2/N for the weights and also controls the degree to which sigmoids are permitted to go into saturation. No significant differences have been found in the learning characteristics of the networks for various values of a in this range. For a more detailed description see [30].

2.3 Learning Methods

The use of artificial neural networks for any useful purpose such as pattern recognition, signal processing, control engineering, etc., requires that the network be trained by a learning algorithm before it can be used. The most popular supervised learning algorithm for non-recursive networks is the error backpropagation algorithm [41, 50]. This training algorithm utilizes a gradient descent technique to minimize an error function. It is a very simple and usually successful method of learning, however its relatively slow learning speed can be a major drawback. Depending on the size and complexity of the network and the nature of the problem, it is not uncommon for backpropagation to require hundreds of presentations of the data set and several days of workstation simulation time to reach a satisfactory result.

There has been much work done on other algorithms and techniques to try to speed up the learning process. One promising area of research for this purpose is the use of second order methods [5, 35]. Backpropagation converges to its solution following the direction of the negative gradient of the error surface at each iteration. However, this may not be the most direct method of convergence and may lead to much wasted learning and relearning. The use of second derivatives within the learning algorithm can eliminate some of this waste and result in much faster learning. Evidence shows that this speed-up can be dramatic in some cases. Speed-up rates of more than 10 have been shown on simple problems [1].

Another area of current research for the purposes of increasing learning speed is the use of novel approaches to learning rate adaptation. Instead of using second order information, the learning rate of stochastic methods can be increased by performing better weight updates at each iteration using only the gradient information. Instead of using a constant learning rate with the gradient information, an adaptive rate can be used to optimize the learning speed using other information acquired from previous updates.

In the following sections, the conventional backprop algorithm, a conjugate gradient descent algorithm using second order information and a methodology for developing and using adaptive learning rates are discussed.

2.3.1 The Error Backpropagation Algorithm

The most popular learning algorithm, due mostly to its simplicity and the fact that it works well, is the "Error Backpropagation" algorithm. Backprop iteratively presents the input patterns to the network and computes the network's output. This output is then compared to the desired output, and the weights of the network are changed as follows:

$$W_i^{(k+1)} = W_i^{(k)} - \mu\delta_i X_i$$

where: $W_i^{(k)}$ is the value of the weight vector in the i^{th} neuron after the k^{th} learning step.

μ is the learning constant ($0 < \mu \ll 1$).

X_i is the input to the i^{th} neuron, including the bias input of 1.

δ_i is the derivative of the squared error for that neuron. For an output layer neuron, $\delta_i = -2(d_i - o_i)F'(s_i)$, where d_i is the desired output of the neuron, and s_i is the result of the scalar product in the neuron. For any other neuron,

$$\delta_i = F'(s_i)\sum_j \delta_j W_{i,j}$$

where: $W_{i,j}$ is the weight connecting the output of the i^{th} neuron to the input of the j^{th} neuron. The summation is over all neurons fed by the output of neuron i.

By iteratively presenting patterns and desired responses to the network, and updating the weights using the backpropagation algorithm, the network learns to correctly classify the patterns in the training set. The network is also able to generalize, and correctly classify many patterns which are not in the training set.

2.3.2 Stochastic Learning versus Batch Processing

Before discussing the new learning methods, we review a fundamental difference between two distinct groups of training methods.

To find the true gradient of the error function with respect to the weights, the error function gradient must be averaged over all of the training patterns. When this calculation is done before the weight update, the method of training is called *batch* mode and a presentation of every training pattern to the neural network is required before each weight update. It is also possible to do a weight update after every individual pattern presentation instead of after the entire set of training patterns. This method of learning is called *on-line* or *stochastic* learning since the patterns are chosen in random order.

There are advantages and disadvantages to both the batch and stochastic methods. Since the total gradient is used in batch mode, the error function is guaranteed to decrease at each weight update.[10] Thus, convergence may be faster, smoother and without many

[10] This is only true if a proper learning rate is used. If the rate is too large, increases in the error are possible.

inaccurate and unproductive learning steps which can occur with stochastic methods. However, stochastic learning can avoid local minima better than batch methods since the steps involve more randomness. This method can also provide a wider search of the weight space and the error surface resulting in a better overall solution. In some situations, for example real time systems, where all the input patterns are not available from the beginning but are presented one at a time, it may not be possible to run learning in batch mode. In the present land mine detection problem, a complete data set is available and both stochastic and batch learning methods are feasible.

Use of the total gradients are required when using second order methods such as the conjugate gradient descent algorithm. However, stochastic methods may be used for adaptive learning rate methods. This must be kept under consideration when comparing the speed of different learning algorithms.

2.3.3 Conjugate Gradient Descent Learning

In the field of optimization, many second order algorithms such as Newton's method, quasi-Newton methods, and conjugate direction methods which offer a much faster convergence rate than standard gradient descent methods have been studied [31, 6]. Many of these algorithms require N^2 storage requirements where N is the dimensionality of the problem, or number of weights in our case. This can impose formidable implementational difficulties. However, one of these methods, the conjugate gradient descent algorithm, requires storage only on the order of N which will make its implementation as a neural network learning algorithm feasible. This is one of the algorithms which was investigated in this work.

Algorithm Derivation

The method of using conjugate gradients is a well-known and studied second order technique of iteratively minimizing a multivariate function [23, 31]. The proper approximations will be made in order to use this method for supervised learning in a neural network. The function which we try to minimize in the neural network is the error function which is the average sum of the squared differences over all the training patterns

$$E = \frac{1}{N}\sum_{p=1}^{N}\sum_{i=1}^{M}[d_{pi} - o_{pi}]^2$$

where d is the desired output, o is the neural network output and i is the output number, p is the pattern number, N is the number of training patterns, and M is the number of neural network outputs. The non-linear mapping function of the neural network can be expressed as a multidimensional Taylor series,

$$f(\mathbf{w}) = f(\mathbf{w}_0) + \sum_i \frac{\partial f}{\partial w_i} w_i + \frac{1}{2}\sum_i \sum_j \frac{\partial^2 f}{\partial w_i \partial w_j} w_i w_j + \cdots$$

where $\mathbf{w}$ is the vector of neural network weights, and $\mathbf{w}_0$ is the origin. If higher order terms are neglected, this function can be approximated by the quadratic equation,

$$f(\mathbf{w}) = c - \mathbf{b}^T\mathbf{w} + \frac{1}{2}\mathbf{w}^T\mathbf{H}\mathbf{w} \tag{1}$$

Here, c is the value of the error function at the origin, $\mathbf{b}$ is the negative of the gradient vector, and $\mathbf{H}$ is the Hessian matrix of partial second derivatives all evaluated at the origin.

To minimize this error function, we can take the derivative of equation 1 and set it equal to zero to obtain

$$\mathbf{H}\mathbf{w}^* = \mathbf{b}$$

where $\mathbf{w}^*$ is the solution weight vector at the function minimum. If $\mathbf{H}$ is invertible, this linear equation can be solved by solving n equations with n unknowns where n is the total number of weights. However, even if $\mathbf{H}$ is invertible, this solution is impractical due to the large number of weights in most neural network problems.

The alternative method of finding $\mathbf{w}^*$ is to use the concept of conjugate directions in an iterative approach. Direction vectors $\mathbf{d}_0, \mathbf{d}_1, \ldots, \mathbf{d}_{n-1}$ are considered $\mathbf{H}$ conjugate if

$$\mathbf{d}_i^T\mathbf{H}\mathbf{d}_j = 0, \ \forall \ i \neq j.$$

In each successive step in the search for $\mathbf{w}^*$, $\mathbf{w}$ will be updated in one of the conjugate directions,

$$\mathbf{w}_{k+1} = \mathbf{w}_k + \alpha_k\mathbf{d}_k \tag{2}$$

for some scalar α_k. Standard steepest descent algorithms such as error backpropagation use a search in the direction of the negative gradient of the error function. Here, we will use the second order information to obtain more intelligent directions of search. The direction vectors $\mathbf{d}_0, \mathbf{d}_1, \ldots, \mathbf{d}_{n-1}$ can be shown to be linearly independent if $\mathbf{H}$ is positive definite. Therefore, it forms a basis of the weight space, and the solution vector can be written as

$$\mathbf{w}^* = \mathbf{w}_0 + \sum_{k=0}^{n-1} \alpha_k\mathbf{d}_k. \tag{3}$$

The problem now becomes finding the α_k's and $\mathbf{d}_k$'s. Multiplying equation 3 by $\mathbf{d}_k^T\mathbf{H}$, substituting $\mathbf{b}$ for $\mathbf{H}\mathbf{w}^*$, and solving for α_k results in

$$\alpha_k = \frac{\mathbf{d}_k^T(\mathbf{b} - \mathbf{H}\mathbf{w}_0)}{\mathbf{d}_k^T\mathbf{H}\mathbf{d}_k}$$

Changing the notation for the gradient vector $\mathbf{g}_k = \nabla f(\mathbf{w}_k) = \mathbf{H}\mathbf{w}_k - \mathbf{b}$, the above equation becomes

$$\alpha_k = \frac{-\mathbf{d}_k^T\mathbf{g}_0}{\mathbf{d}_k^T\mathbf{H}\mathbf{d}_k} \tag{4}$$

Given equation 2 and equation 4, we can find a series of steps which will minimize the error function estimate of equation 1 provided that we can find the set of conjugate

direction vectors $\mathbf{d}_0, \mathbf{d}_1, \ldots, \mathbf{d}_{n-1}$. If we start with the initial negative gradient as the first direction vector, $\mathbf{d}_0 = -\mathbf{g}_0$, each additional direction vector can be found with the recursive relation

$$\mathbf{d}_{k+1} = -\mathbf{g}_{k+1} + \beta_k \mathbf{d}_k \tag{5}$$

Using the conjugate relationship of the direction vectors, we can solve for β_k

$$\mathbf{d}_{k+1}^T \mathbf{H} \mathbf{d}_k = (-\mathbf{g}_{k+1} + \beta_k \mathbf{d}_k)^T \mathbf{H} \mathbf{d}_k = 0$$

$$\beta_k = \frac{\mathbf{g}_{k+1}^T \mathbf{H} \mathbf{d}_k}{\mathbf{d}_k^T \mathbf{H} \mathbf{d}_k} \tag{6}$$

In the above formulation, the Hessian matrix is still required. However, we can eliminate this calculation by recognizing that α_k in equation 2 is the step size for the weight update. We can determine the value of α_k by using a line search to minimize the error function in the $\mathbf{d}_k$ direction. Using this information about α_k, we can eliminate $\mathbf{H}$ by noticing that from equation 2 and the definition of $\mathbf{g}_k$

$$(\mathbf{g}_{k+1} - \mathbf{g}_k) = \mathbf{H}(\mathbf{w}_{k+1} - \mathbf{w}_k) = \alpha_k \mathbf{H} \mathbf{d}_k$$

$$\mathbf{H} \mathbf{d}_k = \frac{(\mathbf{g}_{k+1} - \mathbf{g}_k)}{\alpha_k} \tag{7}$$

Substitution of equation 7 into equation 6 results in the Hestenes-Stiefel formula for β_k

$$\beta_k = \frac{\mathbf{g}_{k+1}(\mathbf{g}_{k+1} - \mathbf{g}_k)}{\mathbf{d}_k(\mathbf{g}_{k+1} - \mathbf{g}_k)} \tag{8}$$

The above analysis provides a method to calculate the solution to the minimization of a quadratic equation in a finite number of steps using equations 2, 5, and 8. However, it should be reiterated that the neural network error function is only approximately quadratic and that the derivatives are estimates of the true gradients so the performance can vary, especially in regions far from the minimum where the quadratic approximation is less accurate.

Derivative Calculation

Although the the mechanics of the conjugate gradient descent algorithm were described in the previous section, some unanswered issues need to be discussed further. These issues involve the actual implementation of the algorithm in software simulation.

The three main equations developed in the previous section used to update the weights of the neural network using the conjugate gradient algorithm for iteration $k+1$ are the following:

$$\mathbf{w}_{k+1} = \mathbf{w}_k + \alpha_k \mathbf{d}_k \tag{9}$$

$$\mathbf{d}_{k+1} = -\mathbf{g}_{k+1} + \beta_k \mathbf{d}_k \tag{10}$$

$$\beta_k = \frac{\mathbf{g}_{k+1}(\mathbf{g}_{k+1} - \mathbf{g}_k)}{\mathbf{d}_k(\mathbf{g}_{k+1} - \mathbf{g}_k)} \tag{11}$$

where $\mathbf{w}_k$ is the weight vector, $\mathbf{g}_k$ is the gradient vector, $\mathbf{d}_k$ is the conjugate direction vector used for weight updating, and α_k and β_k are scalars.

There are several methods of calculating the gradient vector. In our simulations, it is obtained using one backward pass of the error backpropagation algorithm [50, 40]. In terms of computational burden, this method of obtaining the gradient vector is moderately expensive. From the equations in the backpropagation algorithm, the computational complexity of the forward and backward pass is estimated on the order of $O(NN_w)$ and $O(3NN_w)$, respectively, where N is the number of training patterns and N_w is the number of weights. However, in practice we found the backward pass to be about 4.0 to 5.0 times more costly than the forward pass. This may be a motivating factor to investigate other methods of obtaining the gradient vector. One such method would be to use multiple forward paths to estimate the gradient. Despite the computational cost, the current method is most attractive at the moment since the forward pass of the input data through the network naturally produces some of the intermediate values necessary to do the backward pass which can be utilized to reduce the computational burden.

Computation Complexity

Using a standard gradient descent learning algorithm such as backpropagation, the majority of the computational time is spent in the forward and backward pass of the data and the error. When the conjugate gradient descent algorithm is used, there are additional computational requirements which must be addressed. α and β of equations 9, 10, and 11 must also be calculated. Calculating β in equation 11 adds relatively little time. However, to calculate α in equation 9, the current version of the algorithm uses a standard golden section line search algorithm [37]. This line search can be very costly. The simulations run so far indicate that the line search procedure currently used requires between four to five times as much computation time as do the forward and backward pass sections alone. This was true for a wide variety of networks ranging in size from 522 to 2082 weights. These additional computational requirements are very significant. For example, a simulation requiring 10 hours using gradient descent would require between 50 to 60 hours using conjugate gradient with the current line search algorithm to perform the same number of iterations.

The computational burden of the line search is due to the fact that the error of the network, which is calculated by one forward pass through the network for all training patterns, must be calculated several times to determine the weight update step size which will reduce the error to a minimum. Methods to reduce the time necessary for calculating the step size, α, have been investigated. Other line search algorithms are available [31]. However, they would not be able to significantly reduce the computational burden. One very promising technique of avoiding the line search procedure is the scaled conjugate gradient

descent algorithm [33]. Further study on this algorithm may prove to solve many currently existing problems.

2.3.4 Adaptive Learning Rate

In addition to second order methods, an active area of research for increasing the speed of neural network learning is the use of adaptive learning rate parameters for the update of the weights when using stochastic methods. In this section, we will review some of the concepts involving adaptable learning rate parameters and describe one popular algorithm which utilizes these ideas.

Much of the theory involved with learning algorithms for improving performance is neither concrete nor well-established. Most of these algorithms are developed experimentally and are highly problem dependent. However, some sensible heuristics have been established for the purpose of speeding up the neural network learning process [27]. Four of these are:

1. Each weight should have its own individual learning rate.
2. Every learning rate should be allowed to vary over time.
3. When the error derivative of a weight possesses the same sign for several consecutive timesteps, the learning rate for that weight should be increased.
4. When the sign of the derivative of a specific weight alternates, the learning rate for that weight should be decreased.

The first two heuristics are related to the basic assumptions about the weight adaptation learning rates. First, each weight should have its own learning rate since the step size necessary to provide an optimum or even adequate reduction in error will most likely always be different in each weight dimension of the error function. The gradient alone cannot provide sufficient information about the step size which should be used in learning. Often, it can give contradictory information about the step size which should be used. Second, these learning rates should be allowed to vary in time since the weight parameters move around on the error surface which consists of highly irregular shape and curvature.

The last two heuristics are related to the actual adaptation of the learning rate parameters when the first two are assumed to be incorporated. First, if the error derivative with respect to a particular weight repeatedly has the same sign, then the current point is very likely to be on a relatively flat part of the error surface. This would cause a slow descent down this surface in one direction. In this case we would like to increase the learning rate to speed up the descent.

When the sign of the derivative is changing frequently, this is a characteristic of the network oscillating on a portion of the error surface with a high degree of curvature. In this case, it is beneficial to reduce the learning rate so that the error may be reduced to lower levels without wasteful oscillations.

Using these four heuristics for learning rate adaptation, the amount of time spent in the learning phase should be reduced. However, there are countless ways this information could be used to devise a productive algorithm. On method is described in the next section.

Delta-Bar-Delta

The delta-bar-delta algorithm, which adheres to the previously described heuristics for learning rate adaptation, has been proposed [27]. We experimented with this algorithm as a starting point for incorporating learning rate adaptation in the neural network training process.

The weight adaptation for this algorithm is similar to standard error backpropagation. For each weight in the network, the weight update rule is defined as follows:

$$w_{k+1} = w_k - \eta_{k+1} \frac{\partial f}{\partial w_k}, \tag{12}$$

where w_k is an individual weight, η_k is the corresponding variable learning rate, and $\frac{\partial f}{\partial w_k}$ is the partial derivative of the error function with respect to the given weight at iteration $k+1$. In standard backpropagation, η is a non-varying constant vector. But, here we allow η to have a time-varying independent value for each of the weights.

The learning rates, η, follow the adaptation rule given by,

$$\Delta\eta_k = \begin{cases} \kappa & \text{if } \bar{\delta}_{k-1}\delta_k > 0 \\ -\phi\eta_k & \text{if } \bar{\delta}_{k-1}\delta_k < 0 \\ 0 & \text{otherwise} \end{cases} \tag{13}$$

where

$$\delta_k = \frac{\partial f}{\partial w_k}$$

and

$$\bar{\delta}_k = (1-\theta)\delta_k + \theta\bar{\delta}_{k-1}.$$

In the delta-bar-delta algorithm described above, delta (δ) is the partial derivative of the error function with respect to a single weight, and bar-delta ($\bar{\delta}$) is an exponential average of the current and past derivatives. This algorithm adheres to the previous heuristics on learning rate adaptation. When the current derivative of a particular weight and the exponential average of past derivatives of this weight possess the same sign, then the learning rate will be increased by a constant κ. If the current derivative and the exponential average of past derivatives have opposite signs, then the learning rate will be decreased by a factor of ϕ of its current value.

The reasons given by Jacobs for the two different types of adaptation are the following; a linear increment prevents the learning rates from growing too fast, and the exponential decrease allows the weights to decrease rapidly and yet remain positive.

2.4 Practical Considerations

Extensive simulations have shown that second order learning methods have the potential to speed up the training process [13]. However, making a definitive comparison of the learning speed of two different training algorithms can be very difficult. There are many factors, two of which are parameter selection and computational complexity, which affect the performance of a neural network classification system. The implementation method of the algorithm in software can also have a substantial affect on the training speed. Although it is worthwhile to continue investigating new learning algorithms, at this time there is no overwhelming evidence to support using any one algorithm over another. Therefore, most of the simulations using the feedforward sigmoidal neural network conducted for this contract have used the backpropagation algorithm.

3 Probabilistic Neural Networks

An interesting technique with neural application was studied during the later part of this contract period. Instead of using standard neural networks to locate mines, a technique called "Probabilistic Neural Networks" (PNNs) was investigated. The main advantage of this technique is its simplicity. Only one training iteration through the training data is required, and there is only one parameter that can be changed (in the basic model) in order to optimize the mine detection capability. All of the complications associated with neural networks, such as the optimum network topology, learning rate, learning algorithm (etc) are removed. The end performance of the PNN was found to be not quite as good as the performance of the neural network, however it is believed that it is still a useful tool for evaluating novel approaches to preprocessing or postprocessing the data (for example) before testing them with a neural network since there are fewer variables to optimize. The following sections describe the theory behind PNNs. For an excellent discussion on Probabilistic Neural Networks, the paper by Donald Specht [45] is recommended. It will be briefly summarized here.

3.1 Bayesian Classification

PNNs are an architecture for performing Bayesian classification, which is the optimal decision rule when attempting to differentiate patterns. When risk factors[11] are taken into account, this rule is:

$$\begin{aligned} &\text{Decide X belongs to category A when} && h_A I_A f_A(X) > h_B I_B f_B(X) \\ &\text{Decide X belongs to category B when} && h_A I_A f_A(X) < h_B I_B f_B(X) \end{aligned}$$

where $f_A(X)$ and $f_B(X)$ are the probability density functions of categories A and B for an input vector X respectively; I_A and I_B are the penalty functions for making an incorrect decision, and h_A and h_B are the *a priori* probabilities of the occurrence of an item from category A or B.

The boundary line between deciding category A or category B is defined by the equation:

$$\begin{aligned} f_A(X) &= K f_B(X) \\ \text{where} & \\ K &= \frac{h_B I_B}{h_A I_A} \end{aligned}$$

The unknowns which comprise K are the *a priori* probabilities of A and B, which can be estimated from the training data, and the risk function which is user specified. Therefore,

[11] It may be that making an error when classifying patterns is more critical for one type of error than for another. For example, calling a mine pattern "background" is far more critical than calling background "mine". The risk factor is a quantitative measure of this tradeoff.

in order to perform Bayesian pattern classification, we only need to determine (or estimate) the functions $f_A(X)$ and $f_B(X)$. Probabilistic Neural Networks use a Parzen window technique [43][22] to estimate these probability density functions from the training data.

This technique constructs an estimate of the probability density function by summing together multi-variate Gaussian distributions centered at each sample point $X_{A,i}$ in the training data:

$$f_A(X) \approx \frac{1}{(2\pi)^{k/2}\sigma^k}\frac{1}{N_A}\sum_{i=1}^{N_A}\exp\left[-\frac{(X-X_{A,i})^T(X-X_{A,i})}{2\sigma^2}\right]$$

The only parameter to this equation is the "smoothing parameter" σ. Here is an example: Figure 13 shows a two dimensional distribution estimated from eight training points, and uses a small value for σ. Figure 14 shows the same data for a larger value of σ, and Figure 15 shows the same data for an even larger value of σ.

It turns out, that as long as σ is in the right "ballpark", its value is not critical. An example from one PNN program run illustrates this in Figure 16. We see that for σ in the range of about 0.025 to 0.05, the mine detection capability of the network changes very little. Theoretically, as σ goes to zero, the network behaves like a nearest neighbor classifier, and as σ goes to infinity, the decision boundaries become hyperplanes. Typically, for an input vector of high dimensionality, the value of σ should be small.

3.2 PNN Architecture

One of the chief advantages of using the Parzen Window technique for estimating probability density functions is that the resulting Bayesian classifier can be implemented in a neural network like structure. When done this way, it is called a Probabilistic Neural Network. Figure 17 shows the network topology of such a system. There is an input layer, two hidden layers and an output layer of special purpose, neuron-like elements. Each element in the input vector is connected to one input unit whose function is to distribute that value to all of the pattern units on the next layer. There is one pattern unit for each pattern in the training set, and the pattern units are divided up into categories according to the pattern's class (desired response). There is one summation unit on the next layer for each class, and each summation unit adds the responses from pattern units corresponding to its class. Finally, there is an output unit that makes a classification based on the inputs from the summation units.

For the two category "mine versus background" problem, the system is trained by taking each of the input patterns from the training set that correspond to "background", normalizing this input vector by dividing each element by its norm[12] so that the resulting

[12] As defined by Specht, the PNN requires unit norm inputs. However, we have generalized the pattern unit structure in our work to allow any generalized inputs. For our pattern units,

$$F(X,W) = \exp[-(X-W_i)^T(X-W_i)/2\sigma^2]$$

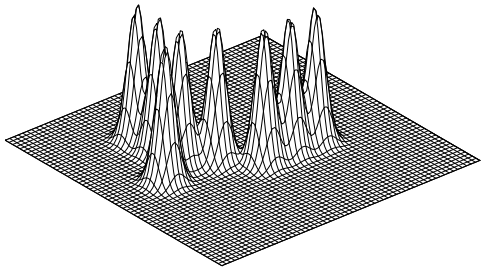


Figure 13: Small Sigma

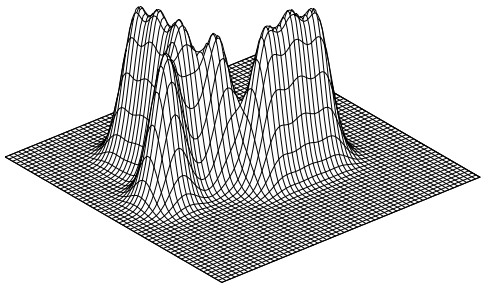


Figure 14: Medium Sigma

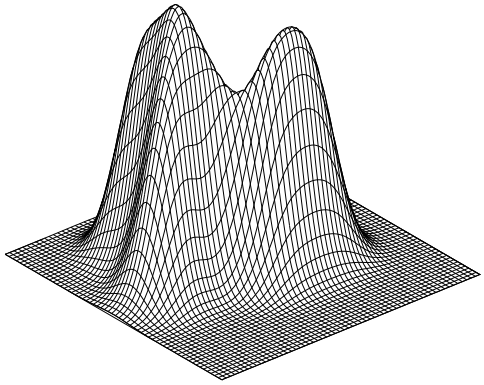


Figure 15: Large Sigma

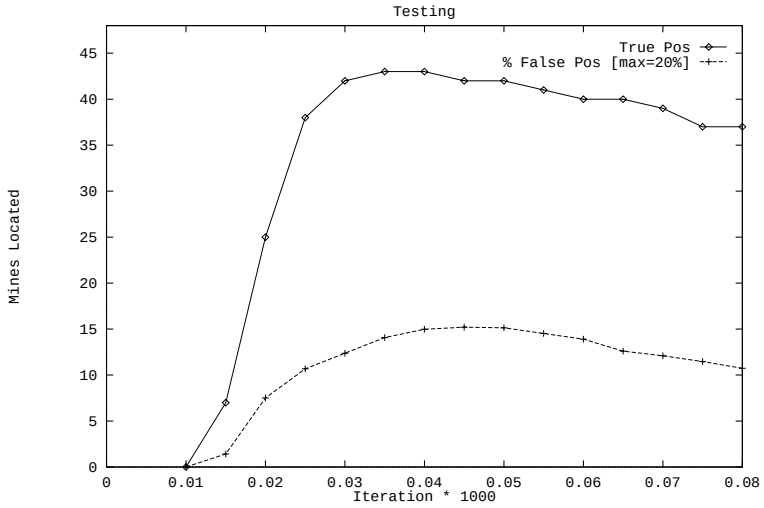


Figure 16: Mines found vary little over a wide range of σ.

norm is 1, and setting the weights of a pattern unit to the components of this normalized vector. Then, this process is repeated for the "mine" patterns, putting their normalized input vectors into the weights of the remaining pattern units. The "background" pattern units are connected to the first summation unit, and the rest of the pattern units are connected to the other summation unit (clearly, this concept is easily extensible to more than two categories, such as when we wish to determine the minetypes rather than simply distinguishing mine from background).

Figure 18 shows the structure of a pattern unit. It is a neuron whose activation function is $\exp[(Z_i - 1)/\sigma^2]$. The function of the pattern unit is to see how "alike" the current input pattern is to the one stored in the pattern unit's weights. A dot product is formed between the input pattern and the weights of the pattern unit[13], and this summation is passed through the nonlinearity. Since we force both the weights of the pattern unit and the input vector to be of unit norm, we see that this forms the exponential function we need in order to form the approximation to the probability density function using the Parzen window technique:

$$\begin{aligned}
\exp\left[-(W_i - X)^t(W_i - X)/2\sigma^2\right] &= \exp\left[-(W_i^tW_i - X^tW_i - W_i^tX + X^tX)/2\sigma^2\right] \\
&= \exp\left[-(\underbrace{\|W_i\|^2}_{1} - 2X^tW_i + \underbrace{\|X\|^2}_{1})/2\sigma^2\right] \\
&= \exp\left[(X^tW_i - 1)/\sigma^2\right] \\
&= \exp\left[(Z_i - 1)/\sigma^2\right]
\end{aligned}$$

[13]Note that the output of this dot product is always maximum, 1, when the input vector is equal to the weights of that pattern unit

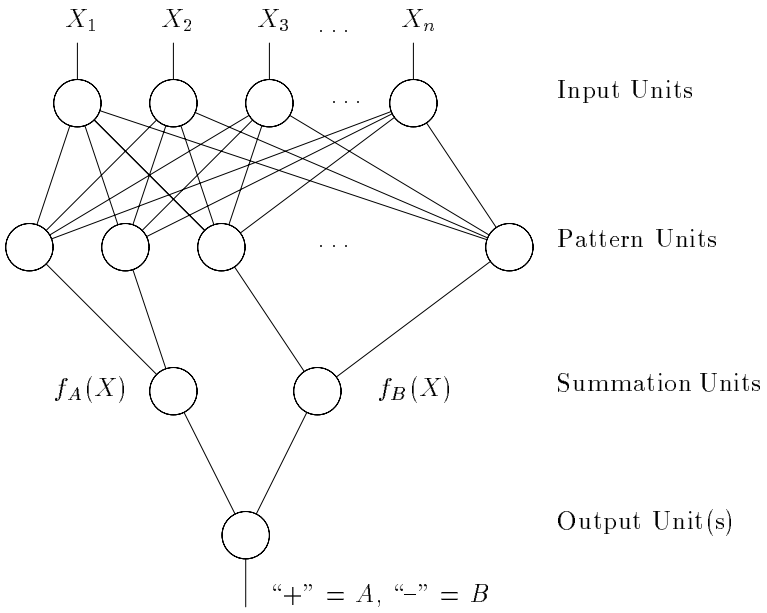


Figure 17: Probabilistic Neural Network Structure

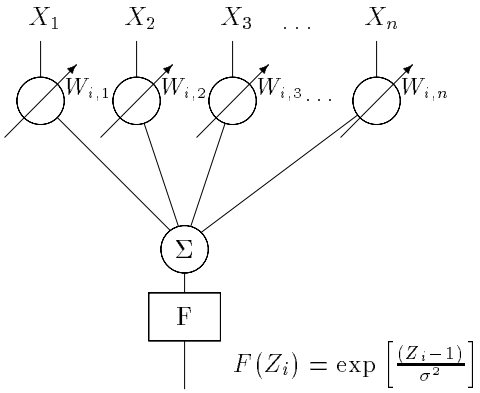


Figure 18: Pattern Unit

The summation unit is a neuron with a linear activation function ($F(Z_i) = Z_i$) and it simply sums the responses of all the pattern units corresponding to its class. The output of the summation unit is then the estimated probability (scaled by a constant) that the input vector X belongs to that category.

The output unit, as shown in Figure 19 is a simple neuron whose activation function is the binary threshold element that makes the decision. The value C in the figure is typically -1, but can take on other values depending on the risk function. $C = -I_B/I_A$[14].

[14]Note: C is not logically related to K from before even though their equations look similar. The constant K includes the values h_A and h_B which cancel out of the analysis when we estimate them from the number of samples of each type in the training data. C is the negative ratio of the risk factors used to bias the decision in the output neuron. For all simulations, except those which emphasized minetypes, $C = -1$.

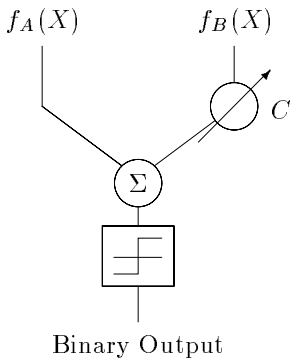


Figure 19: Output Unit

It should be mentioned that although the PNN is very simple, it uses an extravagant number of weights. For 44 inputs/vector and 8343 training patterns, this results in 367 092 pattern unit weights! Methods to reduce the number of necessary training patterns are also available. Specifically, by using any combination of transform or VQ methods, the number of weights can be significantly reduced.

3.3 Nearest Neighbor Classifier

An extremely simple form of the PNN is a nearest neighbor classifier. Such a classifier compares the input vector to all stored input vectors, and determines the stored vector which is closest (in terms of squared error) to the input vector. The output of the NN classifier is then the class of the closest stored vector. As $\sigma \rightarrow 0$ in the PNN, the PNN becomes a nearest neighbor classifier. It should not be expected that the nearest neighbor classifier perform as well as a neural network classifier, or even a PNN classifier since the classification regions in k-space are not as general. Nonetheless, it provides a good lower bound on the performance we would expect from more complex systems. Nearest neighbor classification simulations were performed, and results are listed in Section 9.

4 Preprocessing Methods

While it is possible to train the NN using the raw data supplied by Ft. Belvoir, we have found that by performing some preprocessing operations on the data, the NN results are greatly enhanced. Thus, a wide array of preprocessing techniques has been incorporated into the simulator system. These techniques fall into two categories: global preprocessing and input preprocessing. Global preprocessing is done off-line and requires the entire data set to perform its task. Local preprocessing is done on a vector of globally preprocessed data, and is done on-line during the simulation. The various methods for these tasks are described below.

4.1 Global Preprocessing Methods

The following sections describe two of the global preprocessing methods. Normalization, which is used to "help" the network learn more quickly is discussed first. This is followed by data flattening. Three other major forms of preprocessing used in this study are outlier removal, transform methods, and vector quantization. These subjects are reviewed separately in Sections 6, 7, and 8.

4.1.1 Normalization

Neural networks train fastest when the input to each neuron's activation function is within the linear region of that function. Our NN uses the $\tanh(\cdot)$ activation function, hence we require that the inputs have to be roughly distributed between -1 and 1. If the value of the input to the activation function is far outside this range, the derivative of that activation function $\tanh'(\cdot)$ is near zero, and the weight change is very slow.

Normalizing the input data to have a specific mean and standard deviation is one way to help the network train faster. It is done on a per-data-set basis, where a data set is the measurements for a single S parameter/frequency combination. For each data set $\{x_i\}$, we can compute the mean of the data set to be:

$$\overline{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$$

and the (sample) variance to be[15]:

$$\sigma_x^2 = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \overline{x})^T(x_i - \overline{x}).$$

In order to create a normalized data set $\{y_i\}$ so that it has a new mean $\overline{x'}$ and a new standard deviation $\sigma_{x'}$, we must compute:

$$y_i = \frac{\sigma_{x'}}{\sigma_x}(x_i - \overline{x}) + \overline{x'}.$$

[15]Both the mean and variance are calculated from data in the entire mine lane (as opposed to data from the training region only).

Two normalizations are *always* performed on each data set unless otherwise mentioned. First, the mean is subtracted from the data ($\overline{x'} = 0$), and secondly, each component of each input vector is divided by the standard deviation of that component throughout the mine lane, to force the variance to unity ($\sigma_{x'} = 1$). So, if we combine all of these data sets at each point into a vector, the globally preprocessed vector X_{glob} is related to the mine lane vector X_{raw} as:

$$X_{glob} = Diag\left[\frac{1}{\sigma_1}, \frac{1}{\sigma_2}, \ldots \frac{1}{\sigma_k}\right](X_{raw} - \overline{X}_{raw})$$

Now, consider a neuron on the first hidden layer. It internally needs to compute $f_{net} = W^T \cdot X + b$, where W is the weight vector of that neuron, and b is the bias weight. Consider the set $\{W^*, b^*\}$ which optimizes the weights for the set of input vectors X_{raw}. If, instead of applying X_{raw} to the network, we apply X_{glob}, then,

$$\begin{aligned}
X_{glob} &= Diag\left[\frac{1}{\sigma_1}, \frac{1}{\sigma_2}, \ldots \frac{1}{\sigma_k}\right](X_{raw} - \overline{X}_{raw}) \\
Diag\left[\sigma_1, \sigma_2, \ldots \sigma_k\right] X_{glob} &= X_{raw} - \overline{X}_{raw} \\
Diag\left[\sigma_1, \sigma_2, \ldots \sigma_k\right] X_{glob} + \overline{X}_{raw} &= X_{raw} \\
\text{So, } f_{net} &= W^{*T} X_{raw} + b^* \\
&= \underbrace{W^{*T} Diag\left[\sigma_1, \sigma_2, \ldots \sigma_k\right]}_{=W^{**T}} X_{glob} + \underbrace{W^{*T}\overline{X}_{raw} + b^*}_{=b^{**}} \\
&= W^{**T} X_{glob} + b^{**}
\end{aligned}$$

So, we see that the neural network is capable of finding a new set of optimum weights $\{W^{**}, b^{**}\}$ which provide the identical solution for X_{glob} that $\{W^*, b^*\}$ did for X_{raw}.

This raises the question: "Why perform the normalization in the first place?" Normalization helps keep the sum f_{net} close to zero, (since the weights are initialized to be real numbers with small magnitude) which in turn is close to the linear part of the activation function of the neuron (assuming that a "tanh" like function is being used.) This avoids saturation of the neuron, which is undesirable (especially in the initial stages of learning) since the derivative of the activation function, and hence the weight update will be near zero. Thus, better solutions are more likely to be reached with a normalized input. It also allows the weight vectors to require a much lower dynamic range, which is a very practical advantage when implementing the network. Fewer bits will be required to store weights and input vectors, and the circuitry to perform the internal computations can be smaller.

4.1.2 Data Flattening

One significant deterrent to mine detection was found to be caused by large, step-like jumps in magnitude which periodically occur between successive rows of a data set. Figure 20 shows one example jump in the 800 MHz S_{11} data between rows 414 and 415.

It is thought that these jumps are due to the change in the ambient temperature in the mine lane from one day to the next.[16] As the day progressed, and the temperature rose, the baseline level of response decreased in a ramp-like manner. When the day of collecting measurements concluded and the operator went home[17], the mine lane would cool down. When the operator returned in the morning (or after a week-end) the temperature condition would be significantly different from the evening before, and a jump would occur in the data. It is postulated that, due to the significant slope in the data, even a short lunch break could be the cause of a jump in the data (as evidenced by some jumps "down"). Unfortunately, not all of these jumps are detectable or removable.

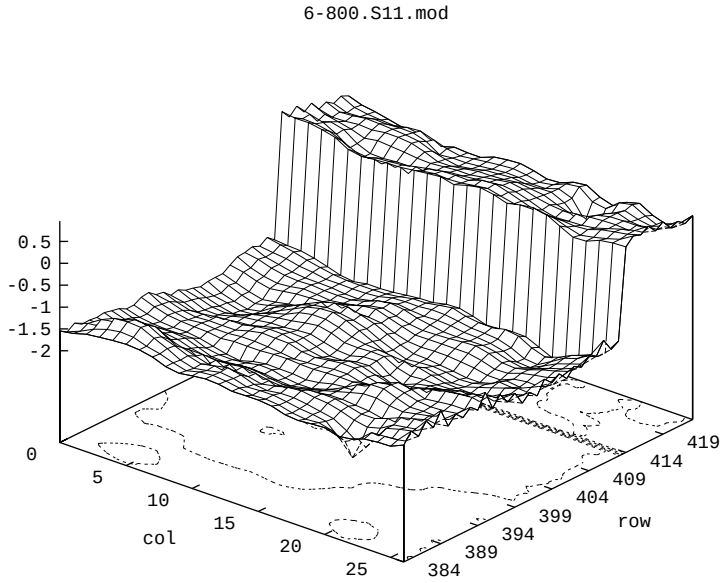


Figure 20: Unflattened Data

We would like to remove these jumps since they are artificial in the sense that they would not occur in a "real" system searching for mines: when searching for mines, one would not pause taking measurements for a significant interval in mid-scan. Furthermore, it is hypothesized that the jumps decrease the network's performance since many different signal levels artificially correspond to mine patterns. Therefore it is desirable to remove the jumps from the data if possible.

A preprocessing function called "Flatten" attempts to eliminate the jumps. It works

[16] Or, perhaps they are caused by a drift over the course of a day in the calibration of the measurement instruments. This is unlikely, though, since if it were true, the jumps would all end on the same "calibrated" level, which does not happen.

[17] Recall that the measurement apparatus automatically measured all data in one row, but required operator assistance to move the apparatus from row to row. This also explains why there are no jumps between columns in a row, but only jumps between successive rows.

as shown in Figure 21

begin {Remove Jumps}

1 Compute the mean and standard deviation of the entire data set.

2 Compute the average over the columns for each row of data.

3 Search for jumps: Tabulate their locations.

4 Calculate the linear regression coefficients for each interval between jumps.

5 Remove the ramp between jumps from the data by subtracting the plane formed from the linear regression coefficients.

end {Remove Jumps}

Figure 21: Algorithm to remove jumps from the data

Once the jumps are found, eliminating them using the last two steps is a straightforward procedure. However, finding the jumps in the first place turns out to be a bit of an "art".

For all methods, the first step is to compute the mean and standard deviation of both the 'x' and 'y' components of the data set. This information is used later when looking for jumps. Next, the average 'x' and 'y' value for each row of data were computed. The idea behind this was that if the difference in the average of either the magnitude (x) or phase (y) between two successive rows was large (in some sense), a jump probably occurred at this location. The "largeness" of a jump is a concept that is relative to the data being considered; however, a specific absolute threshold is needed by the flattening routine to determine if a jump occurred or not. We settled on determining the threshold for jumps by choosing some multiple of the standard deviation of the data.

Three different techniques of increasing sophistication to find jumps were tried. They are described in the following subsections. Since the data collection equipment measured all data within a row automatically, jumps could not occur between columns in a row. Jumps occur between rows because the human operator needed to move the measurement apparatus was not always present when he or she was required.

Simple Jump Search

The first jump removal technique considered each data file separately. For any file, if the average 'x' or 'y' value for consecutive rows differed by more than the threshold value, then a jump was considered to have occurred between those two rows in that data file. The jump was removed as will be described later.

Global Jump Search

This procedure took advantage of some of the information known about how the data was gathered. We know that for every (row, column) coordinate in the mine lane, all

(frequency,S parameter) measurements were taken at the "same time". That is to say, only one measurement pass was taken through the mine lane. Therefore, if a jump was found in one data set (that is, one (frequency,S parameter) set), this implied that a jump also occurred in all other data sets at that row, even if one was not readily apparent.

Therefore, if a jump was found in one data set, then it was considered to have occurred in all data sets, and was removed from the same location in all other data sets.

Global Jump Search with Step Detection

The preceding method found such a large number of jumps that it was considered unuseful. It was postulated that some of the events that were labeled as jumps were actually true high-variance changes in the data. In other words, some of the jumps found were indeed "false jumps". Therefore, an additional step was added to the procedure to be more certain that a jump actually occurred. Once a jump was postulated to have occurred using the previous method, it was verified by checking that all the columns changed value in the same direction across the jump for the data set the jump was located in; they either all increased in value or all decreased in value. All columns in both the 'x' data were required to change in the same direction, and all columns in the 'y' data were required to change in another (not necessarily the same) direction. This addition to the procedure eliminated all of the false jumps. Appendix D shows the 3D mesh plots of the data in the region surrounding each of the located jumps before and after removal of the jump (for a threshold of 1.0 times the standard deviation).

Removing the Jumps

The third jump removal technique was found to work best. The thresholds we used were multiples of the standard deviation of the data; NN results seemed to be best with a threshold of either 1.0 times the standard deviation or 0.0 times the standard deviation (i.e. step detection only).

For all the algorithms, the first step to removing jumps is to compute the average over all columns for the 'x' and 'y' components of each row. Next, the 'x' and 'y' standard deviation of the entire data set is calculated. The previously described jump location procedures are executed. For each interval of data between two jumps, the linear regression of step 4 calculates the best least squares fit of a line to the column averaged data. This is the approximation of the "ramp" experienced during measurement[18]. This line is extended

[18]For further information on the derivations pertaining to how to fit data to a straight line, see [38]. In brief, the procedure works like this: We wish to fit a line, $y(x) = a + bx$ to the data. The coefficients a and b are computed as follows:

$$\begin{aligned} b &= \frac{1}{\sum_{i=1}^{N} [x_i - \bar{x}]^2} \sum_{i=1}^{N} (x_i - \bar{x})\, y_i \\ a &= \bar{y} - \bar{x} b \\ \text{where: } N &= \text{the number of rows between jumps.} \end{aligned}$$

over all of the columns to form a plane. Step 5 subtracts this plane out of the data.

For example, the "SAS6", 800 MHz S_{11} data file has an original average column magnitude as displayed in Figure 22. Many jumps are clearly evident. For example, consider the one between rows 414 and 415 (This is the same jump as in Figure 20).

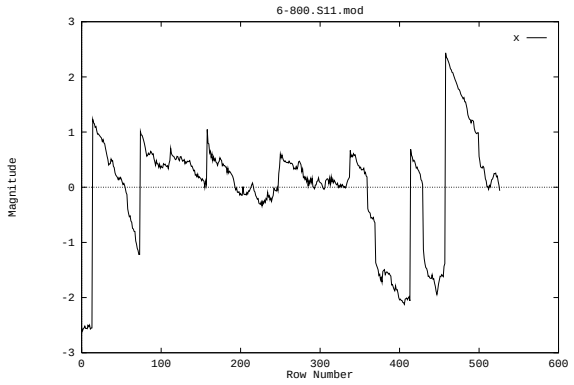


Figure 22: Unflattened Average Row Magnitudes.

When the jumps are removed using this method, the new average row magnitude is as shown in Figure 23a[19]. When this flattened data is re-normalized to have unit variance it is as seen in Figure 23b.

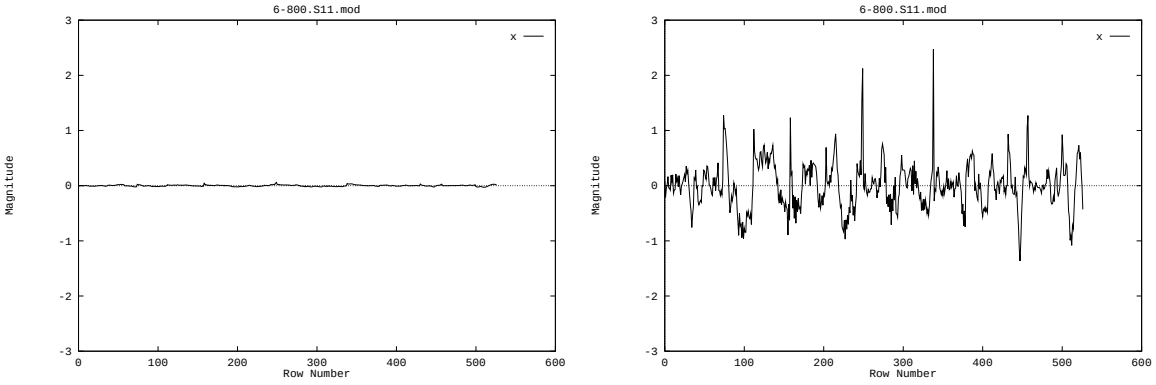


Figure 23: Flattened Row Magnitudes. a) before re-normalization; b) after re-normalization

We see that both the jump and the underlying ramp function is removed from the

x_i = the row number of the i^{th} row.

$\bar{x}$ = $\frac{x_1 + x_N}{2}$

y_i = the average value of row i.

$\bar{y}$ = the average value of the data between jumps.

[19] A threshold of 1.0 standard deviations was used, and jumps were removed at rows: 14, 74, 158, 250, 360, 370, 414, 430, and 458.

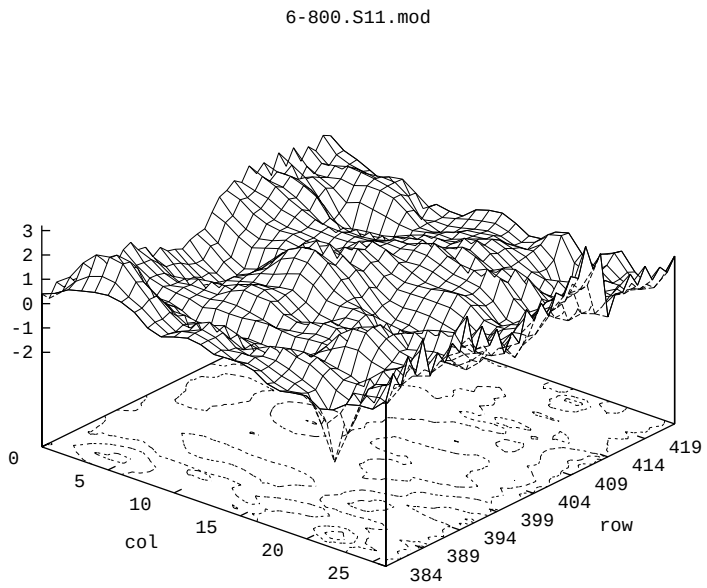


Figure 24: Flattened Data.

row average of the data. Looking at the data in three-dimensional format (Figure 24) we see that the jump has been removed. Several other things can be noticed from Figure 23a. One of them is that while the jumps are clearly better than before, they are not completely removed. This can also be seen in the graphs in Appendix D[20]. Unfortunately there is no real substitute for data that has been collected under identical conditions, but this may be impossible in practice. If temperature is indeed the cause of the ramps, it may be valuable to measure the ground temperature at the time of each measurement. This could be another input to the neural network that may help compensate for the jump effect.

Another thing we note from the re-normalized average row magnitudes in Figure 23b is that the perceptual variance of the data has increased. In actuality, the variance of both data sets is identical since the preprocessor normalized both data sets to have a variance of 1.0, but the "randomness" of the signal is increased. In the unflattened data, the variance has two components: the component due to the sawtooth like ramping functions, and the component due to the variance of the signal. So, $1.0 = \mathrm{Var(Sawtooth)} + \mathrm{Var(Signal)}$[21]. If we remove the sawtooth perfectly, the variance of the signal is therefore increased to 1.0. Since the variance of the sawtooth is quite large, the initial variance of the signal is quite small and the perceptual "randomness" of the data set is small. When the sawtooth is removed, for the same variance, the perceptual "randomness" of the signal is much increased.

[20]The jumps are shown for the 800 MHz data since this data set was most dramatically affected. The frequencies near 1GHz were least affected. This may be one reason why the seven frequency simulations work at least as well as eleven frequency simulations.

[21]Assuming that the sawtooth and signal are uncorrelated.

4.2 Local Preprocessing Methods

The following sections describe the local preprocessing methods. Although, not necessarily a preprocessing method, the first section describes the different types of neural network input vectors used. This is followed by five different forms of local preprocessing.

4.2.1 Neural Network Input Vectors

Initially, the input vector to the NN simulator included data from a "window" of spatially neighboring points. Many different window sizes were tried, and we settled on a standard 7×7 input window. If the center of this window was within 3 grid positions of a mine center, the network was to provide a positive response, otherwise it was to provide a negative response.

Very good results were obtained after several quarters of work using this architecture and some of the preprocessing methods here. Thus, it was determined that we should investigate the more difficult, but also much more practical "minimal" (1×1) window. All recent work has used the minimal window, and thus the mirror/rotate/fold techniques described here are defunct. Their description is included for completeness only. However, should the design ever include a larger input window, these techniques should be re-examined (especially data folding) since they increase generalization and decrease network size.

4.2.2 Gamma Transform

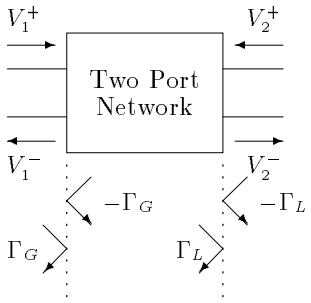


Figure 25: Two Port Network.

The data sets we use from Ft. Belvoir ("SAS6") includes all four S parameters: S_{11}, S_{12}, S_{21}, and S_{22}. To try to incorporate all of this data into a single neural network is possible but a simpler method is desirable. Therefore, an attempt was made to incorporate all of the data into a single more descriptive measure.

The reader is referred to the system topology of a two port network as drawn in Figure 25. It is characterized by the four S parameters: S_{11}, S_{12}, S_{21}, and S_{22}. A single parameter which incorporates the features of all the S parameters is the reflection coefficient, Γ_G.

Γ_G is the reflection coefficient seen at the generating antenna:

$$\begin{aligned}\Gamma_G &= V_1^-/V_1^+ \\ &= S_{11} - \frac{S_{12}S_{21}\Gamma_L}{S_{22}\Gamma_L - 1}\end{aligned}$$

where Γ_L is the reflection coefficient off of the receiving antenna:

$$\begin{aligned}\Gamma_L &= V_2^+/V_2^- \\ -\Gamma_L &= S_{22} - \frac{S_{21}S_{12}(-\Gamma_G)}{S_{11}(-\Gamma_G) - 1} \\ \Gamma_L &= \frac{S_{21}S_{12}\Gamma_G - S_{11}S_{22}\Gamma_G - S_{22}}{S_{11}\Gamma_G + 1}\end{aligned}$$

Solving for Γ_G by substituting in the value for Γ_L,

$$\begin{aligned}\Gamma_G &= \frac{S_{11}S_{22}\Gamma_L - S_{12}S_{21}\Gamma_L - S_{11}}{S_{22}\Gamma_L - 1} \\ &= \frac{(S_{11}S_{22} - S_{12}S_{21})\frac{S_{21}S_{12}\Gamma_G - S_{11}S_{22}\Gamma_G - S_{22}}{S_{11}\Gamma_G + 1} - S_{11}}{S_{22}\frac{S_{21}S_{12}\Gamma_G - S_{11}S_{22}\Gamma_G - S_{22}}{S_{11}\Gamma_G + 1} - 1}\end{aligned}$$

let $X = S_{11}S_{22} - S_{12}S_{21}$

$$\Gamma_G = \frac{X(-X\Gamma_G - S_{22}) - {S_{11}}^2\Gamma_G - S_{11}}{S_{22}(-X\Gamma_G - S_{22}) - S_{11}\Gamma_G - 1}$$

$${\Gamma_G}^2(-S_{22}X - S_{11}) + \Gamma_G({S_{11}}^2 - {S_{22}}^2 + X^2 - 1) + (S_{22}X + S_{11}) = 0$$

Solving the quadratic equation to find Γ_G,

$$\Gamma_G = \frac{({S_{11}}^2 - {S_{22}}^2 + X^2 - 1) \mp \sqrt{({S_{11}}^2 - {S_{22}}^2 + X^2 - 1)^2 + 4 \times (S_{22}X + S_{11})^2}}{2 \times (S_{22}X + S_{11})}$$

Now, we can have a single parameter which incorporates all the S parameters into one. Using the reflection coefficient as the input to the neural network, we can use a single set of complex data and have much of the information of all four complex S parameters. (NOTE: This method was only used for a short time. It was not found to work well – probably because of an over-sensitivity to noisy data).

4.2.3 Rotate and Mirror

The small size and relatively indistinct signal returns of the training data has made mine detection by neural networks difficult. In general, complex problems require neural networks with large numbers of weights. Having a limited amount of training data severely restricts the ability to train such networks.

In order to circumvent these limitations, it would be good to increase the effective size of the training data set. The actual amount of data available is fixed. However, by

taking advantage of several properties of the data and the neural network input system, we can effectively increase the number of patterns in the data set by a large factor. Once we have increased the size of the data set, we should be able to use much larger and more powerful networks. This should then produce more accurate and dependable results. The symmetry properties of the data we have used include rotational invariance and reflectional invariance.

Rotational invariance is the first property of the mine data and neural network input system we used to enhance our data set. A randomly chosen square window of data from the training section of the mine lane is used as the input to train the network for mine detection. If we assume that the data acquisition system has no directional bias, then this window of data has no inherent direction or orientation. Therefore, the orientation of the data window does not matter to the neural network, and this data can be rotated to produce a new set of equally valid data.

For each square window of data, four different inputs can be artificially produced. This can be done by using 0, 90, 180, and 270 degrees of rotation. This corresponds to the signals that would have been obtained if the dirt and mines had been rotated by each of the above amounts. These rotations produce new data for the neural network which was not initially available, but is valid nonetheless. The new rotated data is still the same data taken from the mine lane. It is only the viewpoint or orientation of the network with respect to the mine lane which is changed.

Each of the data windows created by the above rotations is a new input pattern for the neural network, and the corresponding desired response can be determined accordingly. The rotations can be done to each of the windows of data from the entire mine lane. This gives an effective data size which is four times as large as the original data set.

Of course, increasing the amount of training data in this manner is not quite equivalent to having 4 times as much training data, because the noise in rotated versions of the same window is not independent as it would be in four independent measurements. Nonetheless, the additional patterns give us more data to constrain the weights of the network and has a beneficial effect upon classifier performance.

Reflectional invariance is the second property we used to increase the size of the data set. In an analogous manner to the rotational invariance of the data set, the square windows of data also display reflectional invariance.

Assuming symmetric mine shapes, we can flip the data window horizontally or vertically to produce two additional windows of data from each window in the original mine lane. The horizontal and vertical flips correspond to having the mine lane reflected about the center line or scanning the mine lane from the end to the beginning, respectively. This creates the possibility of three different windows of data to be used as the input to the neural network for each data window from the mine lane.

Both rotation and reflection can be used on each data window from the original mine lane to increase the overall size of the data. Using both rotation and reflection and

eliminating redundant transformations, a maximum of eight different orientations of the data can be produced from each data window in the original mine lane. Therefore, using these two methods in conjunction, the effective size of data has been increased by a factor of eight. With this larger amount of data, there is more flexibility with the architecture of the neural network we can use to perform the mine detection. The combined process of rotation and reflection is illustrated in Figure 26.

Some initial simulations showed that there was actually a slight *decrease* in performance due to rotating and mirroring. It was later hypothesized that this performance drop was due to the "ramp" and "jumps" in the input data. Rotating the underlying ramp, and rotating the spurious jumps did not produce plausible generalized input windows, and thus the generalization performance of the entire network was not enhanced. Once the jumps and ramps were removed, the rotate and mirror processing options were found to improve performance.

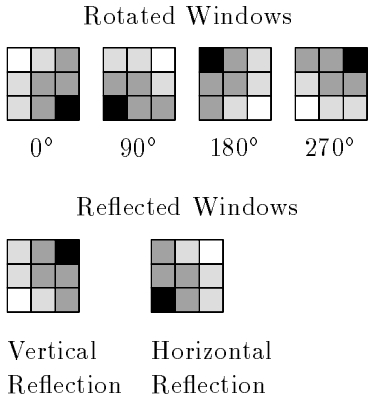


Figure 26: Rotating and Reflecting an Input Pattern.

4.2.4 Data Folding

"Data Folding" is an input preprocessing method which uses input addition to achieve data reduction as well as both rotational and reflectional invariance of the input window. There are two motivations to use this method. First, it enforces rotational and reflectional invariance in the network, and thus should improve generalization. Secondly, as will be shown, it is an alternative method to the Karhunen-Loeve transform to reduce the size of the input vector to the network (although mathematically the two methods are very different)[22]. This allows us to use more input data sets than before, with a network of a given size. Figure 27 illustrates how data folding works.

[22]Part of the appeal of the folding transform is its simplicity. For example, the Karhunen-Loeve transform requires the storage of a $D_i \times D_i$ transform array (where D_i is the dimension of an input window) and requires on the order of $O(D_i^2)$ *multiplications* per input window while data folding requires no such storage, no multiplications, and only $O(D_i)$ *additions* per input window.

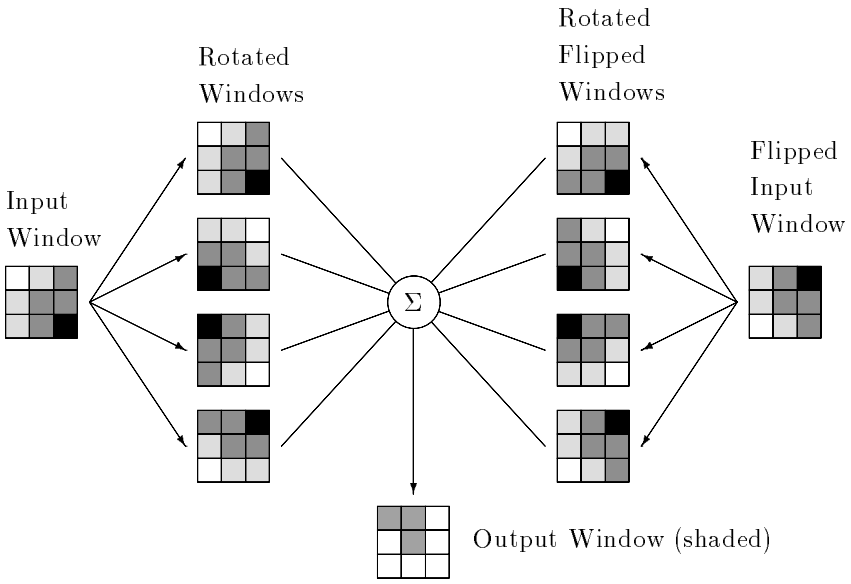


Figure 27: Data Folding

In the figure, the input window is shown to be 3×3 in size for clarity, although data folding is possible with all window sizes[23]. To the right of the input window, the four possible rotations of that window are shown. To the far right, the input window has been flipped across a horizontal axis, and just to its left the four rotations of this window are shown. In these two columns, titled "Rotated Windows" and "Rotated Flipped Windows", we have all eight possible combinations of rotated and mirrored windows. Any other rotation and flip combination will duplicate a member already in this set.

The eight windows are summed together to produce the output window. Because of the rotations and reflections, only three of the values in the resulting window are unique (the shaded entries). These three values are those used to train and test the neural network.

However, to analyze what effect this will have on the network, let us first consider what would happen if not just three, but all components of the output window were used as input to the neural network. No data reduction is accomplished, but the input window will still be the sum of all rotations and reflections. Over time, this will encourage the weights of the neurons in the first hidden layer (which start at random, non-symmetric values) to be symmetric, much like training with separate reflections and rotations of the input window.

By using only the non-redundant components of the output window, we are now performing an operation equivalent to forcing the weights of the first hidden layer neurons to be symmetric. At the same time, we are reducing the number of weights in the first hidden layer due to the reduction in dimension of the input vector. Therefore, both data reduction and data symmetry are performed.

By how much is the input vector reduced? If we consider the minimal window,

[23] However, data folding is meaningless for the minimal (1×1) window.

we see that there will be no reduction. We also observed a three to one reduction in the example just discussed. In general, if D_i is the dimension of the input vector, the dimension of the folded vector D_o is:

$$D_o = \frac{1}{2}\left[\left\lceil\frac{\sqrt{D_i}}{2}\right\rceil^2 + \left\lceil\frac{\sqrt{D_i}}{2}\right\rceil\right]$$

where the $\lceil\cdot\rceil$ operator truncates its operand to the next highest integer. For example, with the frequently used 7×7 window, the output from the folding procedure will have ten elements, to achieve a nearly five to one reduction in data dimensionality. This is probably as big as one would ever want the data window to be but, as a matter of curiosity, we take the limit as D_i goes to infinity and find that the limiting data reduction is eight to one. Also, when D_i is one, D_o is also one, and no data reduction is achieved.

4.2.5 Local Normalization

Local normalization refers to preprocessing which performs a different operation on each input vector X_{glob} from the mine lane, based on the value of that input vector, independent of all other input vectors. Two basic types (either lossy or lossless) of local normalization can be done. These preprocessing techniques were motivated in part by careful theoretical reasoning and in part by curiosity.

Zero Mean

The normalization with the least theoretical basis is the lossy zero mean operation. The function of this is to compute the average of all of the components of X_{glob} and to subtract this average from each component. i.e. $X = X_{glob} - \frac{1}{k}[1, 1, \ldots 1]^T \sum_i X_{glob,i}$. Graphically, we can see this operation as a mapping from a k dimensional space to a $k-1$ dimensional space; a hyperplane in that k dimensional space. In Figure 28, an example with $k = 2$ is shown where all points along the dotted lines are mapped to the solid hyperplane where it intersects that dotted line.

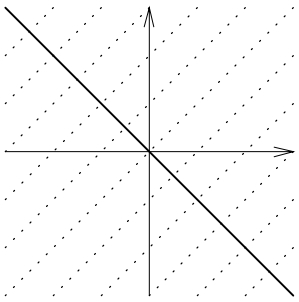


Figure 28: Zero Mean Normalization

Unit Norm

Another normalization method is to divide each vector by its norm, so that it becomes unit norm. i.e. $X = \frac{X_{glob}}{\|X_{glob}\|}$. Graphically, we can see it (Figure 29) as a mapping from a k dimensional space onto a $k-1$ dimensional hypersphere[24] of radius 1 in that space. In the figure, all points on the dotted lines are mapped to the nearest point on the hypersphere which intersect that line.

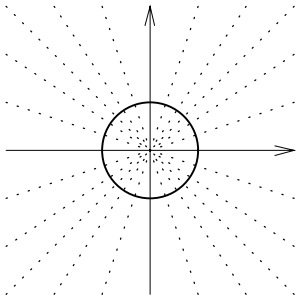


Figure 29: Unit Norm Normalization

If two simplifying assumptions are made, then the Bayes Decision rule can be reduced to a form in terms of unit normalized pattern vectors. The Bayes Rule states:

Choose class i if:

$$\begin{aligned} P(class_i|X) &> P(class_j|X) \quad \forall j \neq i \\ \frac{P(X|class_i)P(class_i)}{P(X)} &> \frac{P(X|class_j)P(class_j)}{P(X)} \quad \forall j \neq i \end{aligned}$$

In the decision rule, $P(X)$ is a constant, so it cancels. If we assume that the *pdf* of X is composed of independently distributed Gaussians with different means, and take the logarithm of both sides, then:

$$\log(\| X - \mu_i \|) - \log(P(class_i)) > \log(\| X - \mu_j \|) - \log(P(class_j))$$

Now, let us assume that the classes are equally likely. Then, the predicted class is

$$\min_i{}^{-1} \| X - \mu_i \|$$

If we divide X and μ by their norms,

$$\min_i{}^{-1} \sqrt{2 - 2\frac{X}{\| X \|} \cdot \frac{\mu_i}{\| \mu_i \|}}$$

or:

$$\max_i{}^{-1} \frac{X}{\| X \|} \cdot \frac{\mu_i}{\| \mu_i \|} = \max_i{}^{-1} \rho_i$$

[24] While the hypersphere itself exists in k-dimensional space, the points on it can be mapped with a non-linear 1:1 mapping to a $k-1$ dimensional space.

where ρ_i is the correlation coefficient. Thus, the pattern recognition problem becomes one of maximizing the correlation between the input vector and the mean vector of each of the possible classes. Geometrically, this becomes choosing the class which makes the smallest angle in k space with the input vector.

The assumptions made are not accurate models of the mine lane data. However the result here seems to work well even when the assumptions are not met. One example is [3] where character recognition based on the correlation principle correctly recognized all but 20 out of 3,064,029 characters from 88 of Sir Arthur Conan Doyle's short stories. It is well known that the character distributions in English text are not uniformly distributed; neither are the vector inputs from the scanner Gaussian (for example, they are strictly positive integer values!) Nonetheless, correlation based pattern recognition works very well here.

It is also interesting to note that the PNN architecture was first defined by Spect based on the correlation principle. In the original architecture, each input vector was unit-normalized. This resulted in significant simplification in the computation in each pattern unit[25].

Both Normalizations

Should both of the normalizations be performed, the result will depend on the order in which the operations are performed. If the order is: zero mean; unit norm, then the result will be a mapping from the k dimensional space to the $k-2$ dimensional space which is a point on the intersection of the hypersphere and the hyperplane. If, however, the order is: unit norm; zero mean, then the result is a mapping from the k dimensional space to a $k-1$ dimensional space. This is illustrated in Figure 30. The starting point X_{glob} is shown as point "1". Should a unit norm be performed, followed by a zero mean, then the point moves from "1" to "2" and ends up at "3". If, however a zero mean is performed, followed by a unit norm, the point moves from "1" to "4" to "5". In all simulations where we perform both normalizations, we first do the zero mean and then the unit norm, thus performing a mapping on a $k-2$ dimensional space.

4.2.6 Approximate Lossless Normalization

In the fear that perhaps some useful information was being discarded with the lossy normalization techniques, a lossless normalization was attempted. It performed the normalization by adding one component to the vector X so that:

$$X = \frac{1}{R_0}\left[X_{glob}^T,\ \sqrt{R_0^2 - \parallel X_{glob} \parallel^2}\right]^T$$

where $R_0 \geq \max \parallel X_{glob} \parallel$.

[25] In our simulations, we use a more general Gaussian function in each pattern unit which does not assume that the input vector has unit norm.

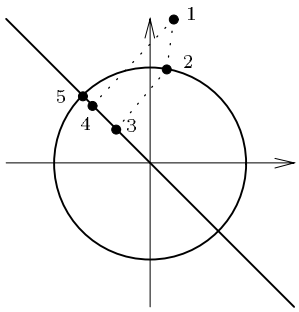


Figure 30: The Order of Normalizations is Important

Thus, each vector X has a scaled version of the input vector X_{glob}, and a normalization coefficient. The only problem is choosing R_0 large enough so that the square root term is positive. From experimentation, it was found that R_0 had to be greater than 52.2.

The problem with needing such a large R_0 is that the normalization coefficient "swamped" the input vector; often becoming the highly predominant component. Thus, an approximate-lossless normalization scheme was devised. In this scheme, R_0 took on a much smaller value. The few components in the mine lane which had a larger magnitude than R_0 were normalized (using the lossy norm) to length 1, and the normalization coefficient was set to 0. Figure 31 shows the relative frequency of vectors in the mine lane with specific norms. This figure considers all of the training data. Looking at the figure, we see that very few vectors have norm greater than 15, so the approximate lossless norm used a parameter R_0 of 15.

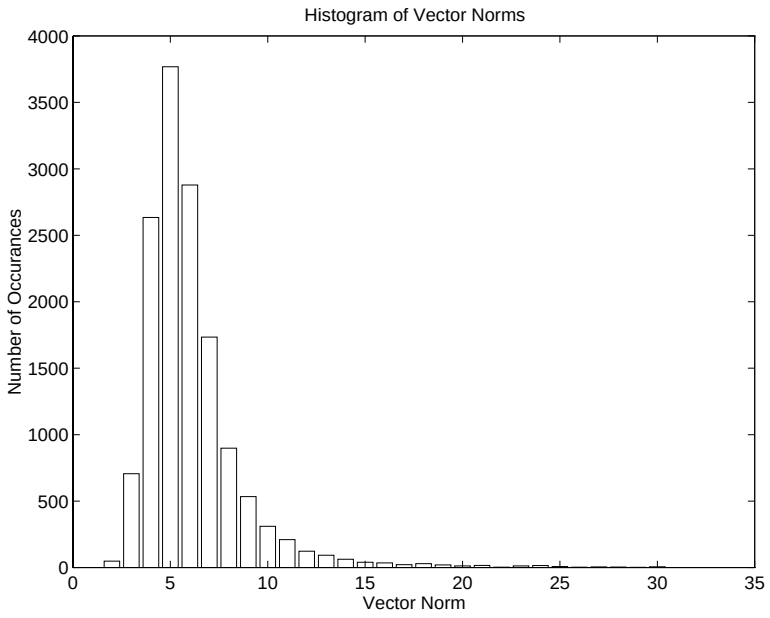


Figure 31: Histogram of Vector Norms of Data

5 Postprocessing Methods

In addition to data preprocessing, some postprocessing functions were also investigated. These were not as rich in variety as the preprocessor since the output of the network is so simple that little can be done with it.

5.1 Number of Outputs

While not specifically a postprocessing operation, the number of outputs from the network forms part of the "back end" of the mine locating process. It is one of the many variable factors to consider when designing a neural network, and choosing it correctly can be important. Some of our simulations have been simulated with two outputs. In this configuration, a separate output was used to determine both mine and background (no mine). This type of binary pattern discrimination can also be accomplished using only one output. If we wished to also predict minetypes, another possible method is to use five or six outputs: one output for each of the five minetypes with or without a background output. Each of these output configurations lends itself to different characteristics in the performance of the network. In addition, each provides a foundation for different types of learning and for postprocessing which can be done. In this section, we will compare the performance of three different types of output structure: a single output, two outputs, and six outputs.

5.1.1 Single Output

Using a single output for the mine discriminating neural network provides the simplest possible output structure for a neural network classifier. In this case, the network is trained to have a high response for mines and a low response for background. The high and low responses can be represented in two different ways depending on the activation function which is used in the neurons of the network. The high value is always '+1' and the low value can be either '0' or '-1' when the neuron's activation function is

$$f(x) = \frac{1}{1 + e^{-\beta x}}$$

or

$$f(x) = \tanh(\beta x)$$

respectively. Both functions behave in a similar manner, and the choice of function used is arbitrary. We have decided to use the later equation in all of the simulations conducted in this report. In this configuration, a +1 output signifies the presence of a mine, and a -1 output signifies the presence of background (no mine).

After the desired response was determined for each position in the mine lane, the network was trained on data from the training sections (three-fifths) of the mine lane. After a predetermined interval of learning, the network weights were frozen and its performance was tested on input data from the test section (the remaining unseen two-fifths) of the mine

lane for cross-validation. Using the single output, there is only one way to interpret the output during the test phase of the simulations. A value greater than a certain threshold indicates a positive point response; a value less than that threshold indicates a negative point response. This threshold value can be adjusted either up or down using information about *a priori* input distributions or output cost functions; however, to do this is less than straightforward and requires more study. For the work presented here, the half-way point, a threshold of zero, was used.

5.1.2 Double Output

Instead of using a single output for mine discrimination, two outputs can also be used to perform the same function. In this case, one output is used to signify a mine and another output is used to signify background. More specifically, a mine desired response is given as a +1 output for the mine output and a -1 for the background output, and the reverse is done for a background desired response. This method adds an additional neuron to the last layer of the network and doubles the number of weights in that layer. This may make it possible to produce more complicated decision regions in the data and improve the overall performance of the network. As a negative effect, it may prevent good generalization by increasing the number of weights in the network.

The use of two neural network outputs can also increase the complexity of the decision to be made about the presence of mine or background. There are many possible methods of using the two outputs to formulate the final decision. The simplest method and the one adopted here is to take the "larger" output (mine output or background output) as the decision. In this situation "larger" is determined by the output closest to +1, which is the same as furthest from -1. This method only utilizes relative information between the two outputs.

Other possible methods could consider the difference between the outputs or the actual magnitude of the outputs. One such method, could involve a minimum difference in the output magnitudes to declare a clear decision. If this criterion is not met, there could be a back up system to determine a proper course of action. For example, if the mine output is 0.6 and the background is 0.2, then there is a difference in the outputs of 0.4. If a minimum required difference of 0.5 had been predetermined, then this mine output could be in question and the prediction could be improved by either testing adjacent positions in the mine lane, or the input data could be altered slightly, by adding noise or by filtering, and tested again. A similar technique could be implemented for having a minimum magnitude requirement for the winning output in order for a safe decision to be made. In either of these two cases, the rule can be set up differently for mine outputs and background outputs to reflect the importance of decisions when considering risk factors. Many other possible manipulations of the two output values can also be conceived.

5.1.3 Multiple Outputs

The final approach to designing the output layer of the network involves the use of a different output neuron for each of the five mine types. In the current study of multioutput networks, six outputs, one for each mine type and one for the background, were actually used. In this type of network, a slightly different desired response which includes information about the mine type of the nearest mine must be calculated. First, the proximity of the nearest mine was determined. If the distance to the center of the mine was within a threshold value, the network output corresponding to this mine was given a +1 desired response. The five other outputs including the background output was given a -1 desired response. In the case where there were no mines within the predetermined distance, the background output was given a +1 output and all others were given a -1. A similar output configuration can also be set up with just five outputs, one for each minetype. In this case, a desired response of all -1's could designate background. However, we are treating background as an equal output to the mines resulting in six outputs.

This type of network which discriminates between different mine types seems the most efficient method to approach the mine detection problem when there are many distinct mine types. Since the different mines have varying sizes and are made of different materials [13], they can produce fairly different sensor outputs. Therefore, in the single or double output case, the network must distinguish all mine signals from background signals. If all the mines produced similar signals which are drastically different from background, the problem could be easily managed. However, if even one or some of the mines produce a signal more similar to background or very different from the rest of the mines, the task becomes much more difficult. Not only is the detection more difficult for the peculiar mine type, but the use of this mine data during the training phase may jeopardize the performance of detecting the other mines. We have seen from work in Section 3 that one particular mine type, mine type number three, has been very difficult to detect. The use of multiple outputs may help reduce this problem.

After the multioutput network has been trained, the outputs of the test region must be interpreted. Similar to Section 5.1.2, there are many ways that this can be done. Initially, we used the simplest method, and use the largest of the outputs as the winner or decision maker. Therefore, whenever any one of the five neurons representing a mine has the largest output, a mine is predicted to be in that position. In order to signify a background, the background output must be larger than all other outputs. For this network, there are several possibilities for the correctness of the output:

1. A mine has been correctly identified by its minetype.
2. Background has been correctly identified.
3. A mine has been incorrectly identified as background.
4. Background has been incorrectly identified as one of the mines.

5. A mine has been incorrectly identified as a different mine type.

Two different measures of performance can be used to evaluate this type of network architecture. In one scheme, only output possibilities 1 and 2 are assumed to be correct and all others are classified as errors. This would determine how well the network can identify and classify the different types of mines. However, this is a little too stringent for our initial use of the multioutput network, since misclassifying a mine as in output possibility 5 above is not a fatal error. For the purposes of mine detection in this contract, we grouped all mine outputs of the network as one type and used the grouping postprocessor (Section 5.2) to determine the mine positions. In future study, we will look at the possibility of discriminating mines of different type with the use of different postprocessing methods.

Figure 32 shows how our network diagrams depict the multi-output capabilities of the simulator. From left to right we see output sections for one, two and six output networks.

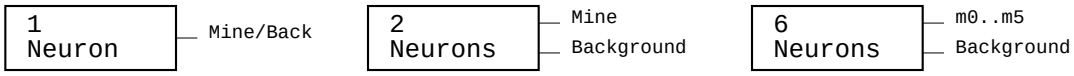


Figure 32: Schematic Symbols for 1, 2 and 6 Output Networks

5.2 Grouping Mines

A previous method of reporting simulation results worked by grouping together network point responses. During the test phase in which the weights are held constant, the input window is scanned over the entire mine lane. At each spatial location, the point response of the neural network is tabulated. Figure 33 represents an example of the raw quantized network output. An 'M' value indicates a positive point response from the neural network and a 'b' indicates a negative point response. Without further processing, the information in this table is hard to interpret. What we really want is to know how many mines have been found, and how many false positives there were. This is accomplished by post-processing this table of 'M's and 'b's to extract the positions where the network thought there were mines.

The grouping algorithm looks for contiguous areas of 'M's. Any region that contains 3 or more 'M' values joined together was considered to be a mine, and any other region of fewer than 3 'M' values is discarded as noise. For each region, the centroid of the 'M' positions is calculated. A sample joined region is shaded in the figure. All other regions with 'M' in them are not shaded since they were too small to be considered a mine.

Once the program has found a list of these regions and their centroids, it matches it against the list of 'true' values that are the known locations of the mines. This information is found in in Appendix A. If the centroid of a positive region is within 4 grid units of a

mine's center, it is considered to be a true positive (a mine was really there). Otherwise, it is considered to be a false positive (no mine was there).

b	b	b	b	b	b	b	b	b	b	b
b	b	b	b	b	b	b	b	b	**M**	b
b	b	b	b	**M**	b	**M**	b	b	b	b
b	b	b	b	**M**	**M**	b	b	b	b	b
b	**M**	b	b	b	**M**	**M**	b	b	b	b
b	**M**	b	b	b	b	b	b	b	b	b
b	b	b	b	b	b	b	b	b	b	b

Figure 33: Postprocessing by Grouping Network Responses

While we have abandoned this method for a simpler way to report results, it holds merit if postprocessing is an implementation option.

5.3 Filtering

One of the methods discovered while investigating PNNs (see Section 3) was output filtering. The idea behind filtering is to remove spurious network responses by combining them in a weighted manner with adjacent responses.

During the test phase of the neural network, tables are created with the network response at every location in the mine lane. There is one table for every output neuron. Spatial filtering can be performed on these tables; this is called Pre Quantizer filtering. The information from these tables is then combined to form a single table of point decisions. The entries of this table are the '+1/-1' decisions based on the rules in section 5.1. This is shown in Figure 33 as 'M' for +1 and 'b' for -1. This table can be filtered as well; this is called Post Quantizer filtering. Finally, the response grouping subroutine is executed on this table to give a list of predicted mine locations. This process is illustrated in Figure 34.

The filter we use is a 3×3 filter shown in Figure 35. The filtering operation replaces each point response with k^2 times itself plus k times the sum of the horizontal and vertical neighbors plus one times the sum of the diagonal neighbors. (Edges are handled by treating out-of-mine-lane responses as zeros). The choice of filter is somewhat arbitrary, but it does have the feature that it has a (scaled) sampled Gaussian form[26]. Consider the scaled two dimensional Gaussian:

$$\mathrm{fn}(d)|_{d^2=x^2+y^2} = k^2 \exp\left(\frac{-d^2}{2\sigma^2}\right)$$

[26]The scaling doesn't matter since we are using +1 and -1 as network responses, and decisions are based only on the output sign, not on its magnitude. If we were using +1 and 0, the filter would have to be normalized by dividing each element by $4 + 4k + k^2$ or the decision threshold would need to be multiplied by this value.

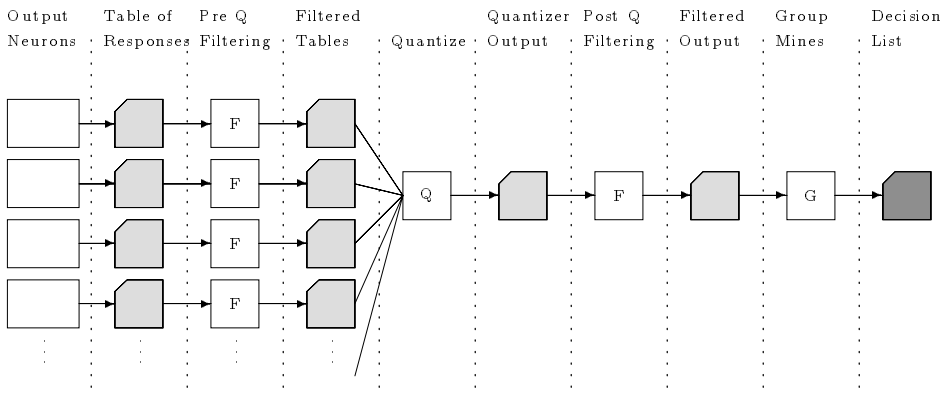


Figure 34: Postprocessing Stages with Filters

1	k	1
k	k^2	k
1	k	1

Figure 35: 3×3 Spatial Filter

Suppose we let:

$$\sigma^2 = \frac{1}{2\ln(k)}$$

then:

$$\begin{aligned} \mathrm{fn}(d) &= k^2 \exp\left(-d^2 \ln(k)\right) \\ &= k^2 \exp\left(\ln(k^{-d^2})\right) \\ &= k^2 k^{-d^2} \\ &= k^{2-d^2} \end{aligned}$$

At $d = 0$ (the center tap), the value is k^2. At $d = 1$, the value is k, and at $d = \sqrt{2}$, the value is 1. This confirms the filter shown previously. Some thought shows that a high value of k emphasizes the center tap, and a low value of k provides more smoothing during the filtering. The value most frequently used was $k = 2$, but some simulations were performed with other values, most notably $k = 3$. As $k \to \infty$, there is no filtering. As $k \to 1$ the filtering is maximally smooth. (For $k < 1$, $\sigma^2 < 0$ which is illegal).

The simulator can optionally: not filter at all; only pre quantizer filter; only post quantizer filter; or pre and post quantizer filter. From the simulations, we found that filtering tends to reduce both the number of true positives and false positives.

6 Outlier Removal

The effectiveness of training a neural network is affected greatly by the training data set which is used. No learning algorithm will work effectively if the data used does not contain sufficient information presented in a clear and unambiguous manner. This can be a problem which is not easily detectable, especially when using very high dimensional data. Since the data is impossible to visualize, it can be difficult to detect any obvious anomalies which may interfere with the learning process of the neural network. This is the case for our mine lane data.

Due to the nature of the mine data, it is not unreasonable to expect some anomalous data points which would hinder the learning and generalization abilities of a neural network. Therefore, the removal of these outlier data points from the training set should increase the overall performance of the network. The presence of inconsistent data points can be attributed to uneven soil conditions, including moisture and debris, inconsistent depth of mines, sampling of data over mine edges, inaccurate measurement of exact mine positions, and some other causes. In any case, the detection and removal of these data points from the training set has been emphasized in the last year of the research contract.

One of the main reasons for the presence of undesirable data points in the training data is due to the collection of data at or near the edges of the mines in the mine lane. These areas may be producing signals which cannot be classified clearly as either mine or background. These ambiguous signals may interfere with the learning and classification abilities of the other distinct signals. Therefore, the first attempt at outlier removal has concentrated on the detection and removal of data points which are near the mine edges. Through collaboration with Dr. Don Torrieri at the Army Research Laboratory in Adelphi, Maryland, a new data preprocessing method has been developed to perform this simple type of outlier removal. This procedure uses simple heuristics based on the physical geometry involved in the signal collection procedure. Other methods of outlier removal have also been studied and are presented later in this report.

Data collected near mine edges usually have neighboring points which do not have the same desired response. Therefore, two neighboring points with somewhat similar signals may have opposite desired responses. This may confuse the network. Both of those data points should not be used in the training data set. Following this reasoning, only data points which are in the interior of a mine or the background region should be used for training. In order to insure this condition, all the neighboring data points should be checked to make sure that an included point is surrounded only by other data points with the same desired response. This was done by checking the four nearest neighbors of each data point. A data point was included in the training set only if all of its four neighbors had the same desired response. Along the edges and corners of the mine lane, the missing neighbor was excluded from the requirement.

As an extension of the four nearest neighbor outlier removal method, an eight nearest

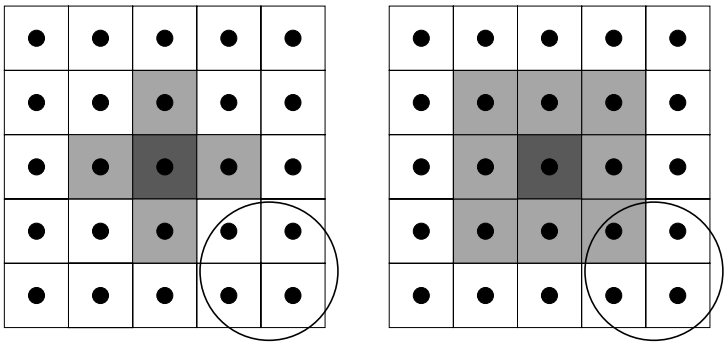


Figure 36: Four and eight nearest neighbor outlier removal.

neighbor outlier removal method has been implemented. This method examines the eight neighbors of each data point, and if all of them do not have the same desired response as the center point, it is excluded from the training set. Figure 36 shows an example of the two methods. In this figure, the dark shaded center point is a background data point in question and the circle is a nearby land mine. In (a), the mine does not cover any of the four nearest neighbors and the center point would be included in the training set as a background data point. However, in (b), one of the eight neighbors is over the mine, so this point would be removed from the training data set. As demonstrated by this example, using the eight nearest neighbor outlier removal method instead of the four nearest neighbor method has the effect of removing several additional data points from the training data set.

Initial use of the four nearest neighbor outlier removal method showed diminished mine detection rates [19]. Inspection of the outlier removal technique and its effect on the training data set showed that the distribution of mine data and background data after outlier removal had become very skewed. The original training data sets had approximately an 7:1 ratio of mine to background data. After the outlier removal was applied the ratio became approximately 20:1. The large bias toward background data could be one explanation for the very low detection rates produced. A new training system was implemented in an attempt to try to mitigate the ill effects produced from training the neural network using the unevenly distributed data set.

6.1 Data Balancing

Since a heavily unbalanced training data set will produce a bias in the outputs of the network, it was hypothesized that it may beneficial to try to equalize the distribution of mine and background outputs in the training data. To achieve this balance, the training was done using the stochastic backprop method and by presenting both mine and background data patterns with approximately the same frequency. This resulted in each of the mine data points being used in the training of the neural network more frequently than any of the background data. This does not provide a perfect solution for balanced training since the

network will be trained on a small set of data more frequently than the others. Although, this may not be the solution for training the neural network for optimum generalization performance, it was an attempt to improve the performance of the heavily uneven data set created by using outlier removal.

Comparing the results after data balancing the training set against the results without data balancing, a remarkable increase in the mine detection performance was observed. This was very promising and may have explained one of the reasons why the four nearest neighbor method was showing a decreased mine detection rate.

6.2 Modified Outlier Removal

The two problems using the outlier removal method seems to be that the distribution of mine data and background data becomes very uneven and the total number of data points representing mines also becomes too small. The use of data balancing when training the network, although not a perfect solution, seems to alleviate some of the problems with the uneven data distribution. However, the very limited number of mine exemplars does limit the effectiveness of the neural network to generalize over a larger input data space. Therefore, in an attempt to get the benefits of outlier removal and still maintain as many different mine data inputs as possible, a new modified outlier mechanism was developed. Instead of removing all data points which do not satisfy the four or eight nearest neighbor criterion, only the background data are subjected to outlier removal. Therefore, all of the mine data is retained in an attempt to maintain a large data pool of mine data and the outlier removal procedure was conducted only on the background data. The background data near mines are removed from the training set. Although this method retains the mine data at the edges of the mine, the hope is that the benefit from having a larger mine data set will be greater than the detriment of having some outlier mine data. The mine/background data balancing was also always done on the modified outlier removed training data sets. The results from using the modified outlier removal method and data balancing were very favorable.

7 Transform Methods

A number of similar but different "transform" data preprocessing methods were investigated during this contract period. All of these transforms have the property that they form a vector Y,

$$Y = A^T X$$

where A is the transformation matrix, and X is a k dimensional preprocessed vector of data from the mine lane.

The motivation behind this transformation is twofold. First, the pattern space may be transformed in such a way that a PNN or an NN is more capable of separating the mine patterns from the background patterns. Secondly, data reduction can be achieved: i.e. we can compute $Y_m = A_m^T X$ where A_m contains the first m of the k columns of A, and Y_m has the resulting dimension m. Generally, better separation and lower dimensionality can be attained at the same time, and the data reduction is achieved without a significant loss of "energy" from the input data, and thus without a significant loss of performance. This data reduction will result in faster training times and faster and more compact network realizations. Figure 37 shows the data flow block diagram used by all the transformation methods. The globally preprocessed data from the minelane, X_{glob}, is preprocessed to form X, which is transformed to form: $Y_m = A_m^T X$. The vector Y_m is then input to either an NN or a PNN as appropriate, and training and testing is performed as usual.

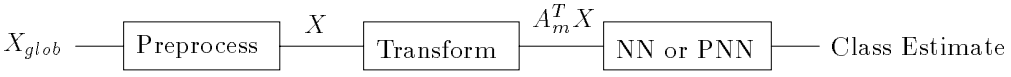


Figure 37: Data Flow in the Transformation

7.1 The Karhunen-Loeve (KL) Transform

The Karhunen-Loeve Transform [43] will be discussed first since it forms a basis for discussing the more complex methods which follow.

The KL Transform is the only "unsupervised" transform that we investigated. The notation "unsupervised" refers to the fact that the transform matrix A was generated from the training patterns X in the mine lane, without knowing what the classes of the training patterns were.

The KL method (otherwise known as principal components analysis, or the Hotelling transform) performs a linear transformation on the input data. The input vector X is multiplied using an orthonormal matrix A to form $Y = A^T X$. Since A is orthonormal, the resulting operation is a rotation of the input data in k-space.

We wish to compress the information content of the input window into as few parameters as possible. To see that this can be done, imagine a random variable Z whose

probability density function (*pdf*) resides completely on some 2-D plane in 3-space. Three parameters are required to specify any of the values that the random variable Z can assume. However, suppose we rotated the entire space by some transformation such that the rotated *pdf* lies on the x-y plane. In this new coordinate system, only two parameters are required to specify any point in the *pdf*.

In the mine-detection problem, if all S parameters and all frequencies are being used, then there are k=44 input coefficients to the neural network (88 if phase values are being used too). Assuming that the relevant information content required for finding mines lies in some sub-space of the 44-D space, we can find a matrix A_m to rotate and compress the input window such that we can extract only the parameters we need. We choose the matrix A which performs a 44-dimensional rotation so that any input window can be specified with the fewest number of values. The correct choice of this matrix is the modal matrix of X which can be found as follows:

First, compute the covariance matrix for X.

$$\Sigma_X = E[(X - \overline{X})(X - \overline{X})^T]$$

Then, compute the eigenvalues (λ_i) and eigenvectors (u_i) of Σ_X. Since Σ_X is positive semi-definite, all the eigenvalues will be non-negative and real. Without loss of generality, we can order the eigenvalues such that

$$\lambda_1 \geq \lambda_2 \geq \cdots \lambda_k \geq 0.$$

Compose the matrix A by setting its columns equal to the ordered eigenvectors:

$$A = [u_1 u_2 \cdots u_k]$$

Now, let's consider the ramifications of representing feature vector X using $m < k$ components of Y. Since X and Y are random vectors, the meaningful measure of error is an expectation, and in particular the mean square error metric works well. First, make some definitions:

$$A = [A_m : A_k]$$

where A_m is a $k \times m$ matrix consisting of the first m columns of A (the first m eigenvectors of Σ_X) and A_k consists of the remaining $(k - m)$ deleted columns of A, corresponding to the $k - m$ terms not used to represent the vector X. Thus,

$$X = [A_m : A_k] \begin{bmatrix} Y_m \\ Y_k \end{bmatrix}$$

where Y_m is the vector consisting of the m components of Y used to represent X; Y_k consists of $k - m$ components not used. The representation of X, denoted as $\hat{X}$ resulting from using only m elements of Y may be written as:

$$\hat{X} = A_m Y_m$$

Therefore the representation error is given by the $k \times 1$ vector ϵ where:

$$\begin{aligned} \epsilon &= X - \hat{X} \\ &= A_k Y_k \\ &= \sum_{i=m+1}^{k} a_i Y_i \end{aligned}$$

where a_i is the i^{th} column of A, and Y_i is the i^{th} component of Y.

With some math, this reduces to:

$$E[\| \epsilon \|^2] = \sum_{i=m+1}^{k} \lambda_i$$

Clearly, if we are to minimize the error while discarding components from Y, we must discard those components with the smallest eigenvalues.

7.1.1 Application of the KL Transform

In order to use the KL transform, one must first compute the modal matrix A. This was done as follows: The input mine lane was broken up along its length into five sections. The first, third and fifth sections were considered to be training data, and the remaining sections were testing data (This is consistent with the rest of the report). A covariance matrix Σ_X was computed by taking all possible input vectors X from the training set and computing[27].

$$\Sigma_X = \frac{1}{N-1} \sum_{i=1}^{N} (X_i - \overline{X})(X_i - \overline{X})^T$$

To compute this with one pass through the data (to compute $\overline{X}$ and Σ_X in parallel):

$$\begin{aligned} \Sigma_X &= \frac{1}{N-1} \sum_{i=1}^{N} (X_i - \overline{X})(X_i - \overline{X})^T \\ &= \frac{1}{N-1} \sum_{i=1}^{N} \left(X_i X_i^T - X_i \overline{X}^T - \overline{X} X_i^T + \overline{X}\,\overline{X}^T \right) \end{aligned}$$

[27] If the value of $\overline{X}$ were known explicitly, then we would need to calculate $\Sigma_X = \frac{1}{N} \sum_{i=1}^{N} (X_i - \overline{X})(X_i - \overline{X})^T$. The expected value of this calculation is: $E[\Sigma_X] = E[X^2] - \overline{X}^2$, which is an unbiased estimator of the covariance array. However, we do not know the value $\overline{X}$, and must estimate it as: $\frac{1}{N} \sum_{i=1}^{N} X_i$. With some calculation, we can show that it is necessary to use the multiplicative constant $\frac{1}{N-1}$ when calculating Σ_X in this case in order to form an unbiased estimate.

This issue is relatively unimportant, since we are most concerned with the eigenvectors of Σ_X, which are unaffected by the constant. Each of the eigenvalues will be different by a factor of $\frac{N-1}{N}$ if the "incorrect" scheme is used. The eigenvalues are only used to calculate the number of eigenvectors used to form A_m, and the factor $\frac{N-1}{N}$ is so close to 1, that there will probably be no difference in the decision whichever selection is made.

$$
\begin{aligned}
&= \frac{1}{N-1}\left[\left(\sum_{i=1}^{N} X_i X_i^T\right) - \left(\sum_{i=1}^{N} X_i\right)\overline{X}^T - \overline{X}\left(\sum_{i=1}^{N} X_i^T\right) + N\overline{X}\,\overline{X}^T\right] \\
&\text{Let } \Phi = \textstyle\sum_{i=1}^{N} X_i X_i^T \text{ and } \phi = \sum_{i=1}^{N} X_i. \text{ Then,} \\
\Sigma_X &= \frac{1}{N-1}\left[\Phi - \left(N\overline{X}\right)\overline{X}^T - \overline{X}\left(N\overline{X}^T\right) + N\overline{X}\,\overline{X}^T\right] \\
&= \frac{1}{N-1}\left[\Phi - N\overline{X}\,\overline{X}^T\right] \\
&= \frac{1}{N-1}\left[\Phi - \frac{\phi\phi^T}{N}\right]
\end{aligned}
$$

where Φ and ϕ can be easily computed in parallel in a summation loop. The eigenvalues and eigenvectors of Σ_X are computed and the matrix A_{KL}^T is generated and stored. This is done as an off-line procedure, although methods exist for computing it iteratively using a neural-net like structure. For a full algorithm to generate the A_{KL} matrix, see Figure 38.

begin {generate A_{KL}}

$N = 0$, $\phi = [0]_{k\times 1}$, $\Phi = [0]_{k\times k}$.

For each preprocessed vector X_i in the training section:

If X_i is not an outlier, then:

$\phi \leftarrow \phi + X_i$

$\Phi \leftarrow \Phi + X_i X_i^T$

$N \leftarrow N + 1$

endif

endfor

$\Sigma_X = \frac{1}{N-1}\left[\Phi - \frac{\phi\phi^T}{N}\right]$

Compute the eigenvectors u_i and the eigenvalues λ_i of Σ_X.

$A_{KL} = [u_1 u_2 \ldots u_k]$, sorted by descending λ_i.

$b_{KL} = [\lambda_1 \lambda_2 \ldots \lambda_k]^T$

Store A_{KL}^T, b_{KL} for later use.

end {generate A_{KL}}

Figure 38: Algorithm for generating A_{KL}

To perform the KL transform, we create the matrix A_m from A_{KL} as:

$$A_m = A_{KL}[1:m].$$

That is, A_m is a $k \times m$ dimensional array which holds the first m columns of A_{KL}. The value of m is chosen to meet the energy criterion:

$$m = \min_j{}^{-1}\left[\frac{\sum_{i=1}^{j}\lambda_i}{\sum_{i=1}^{k}\lambda_i} \geq \frac{E}{100}\right], \quad 0 \leq E \leq 100$$

So, A_m holds the minimum number of eigenvectors required to maintain at least E percent of the energy in the transform. Once A_m is formed, the transform is carried out as outlined above: For each input vector X,

$$Y_m = A_m^T X$$

This vector is used as input to either the NN or PNN, as appropriate, and training and testing is performed as usual.

7.1.2 Whitening the KL Transform

The KL transform is an orthonormal transform. That is, it preserves the structure of the covariance matrix, and Euclidean distances between points are maintained. Only rotations can be performed.

We could also, instead of composing A_{KL} with the eigenvectors of Σ_X, form $A_{KL,w} = [u_1' u_2' \ldots u_k']$ where $u_i' = \frac{u_i}{\sqrt{\lambda_i}}$. Then, we are computing a whitening transform which first rotates the input space, and then makes the variance along each dimension equal to one. This can result in simplifying the decision boundary by "flattening" it out. Additionally, since the scale of each transform coefficient Y_i is normalized, then the network will be more easily implemented with fixed precision weights. There is also reason to believe that network training may become faster since the NN training method is based on steepest descent learning which in turn converges most quickly for low eigenvalue spread. Whitening is a lossless operation, in the sense that a NN can reach exactly the same solution as with non-whitened data by replacing the first hidden layer neuron weights W by $Diag[\sqrt{\lambda_i}]W$.

7.2 The KL+ Transform

Since we know the class of each pattern in the training region, it seems logical that we can use this knowledge to our advantage. Suppose we were to compute the transform matrix A_m using only patterns corresponding to mines.

Then, the input space would be rotated so that the covariance matrix of the mine patterns would be most efficiently spanned by the basis vectors. The regular KL transform would make the entire combined covariance matrix be most efficiently spanned, and perhaps would not function as well. This is illustrated in Figure 39.

The algorithms for computing A_{KL_+} (and A_{KL_-}, used for the "Composite Transform") are shown in Figure 40. As with KL, A_m is formed from A_{KL_+} to meet the energy criterion, and $Y_m = A_m^T X$.

7.3 The Composite Transform

The KL+ transform tries to take into account the covariance matrix corresponding to patterns where mines are present. However, improvement may still be achieved by considering

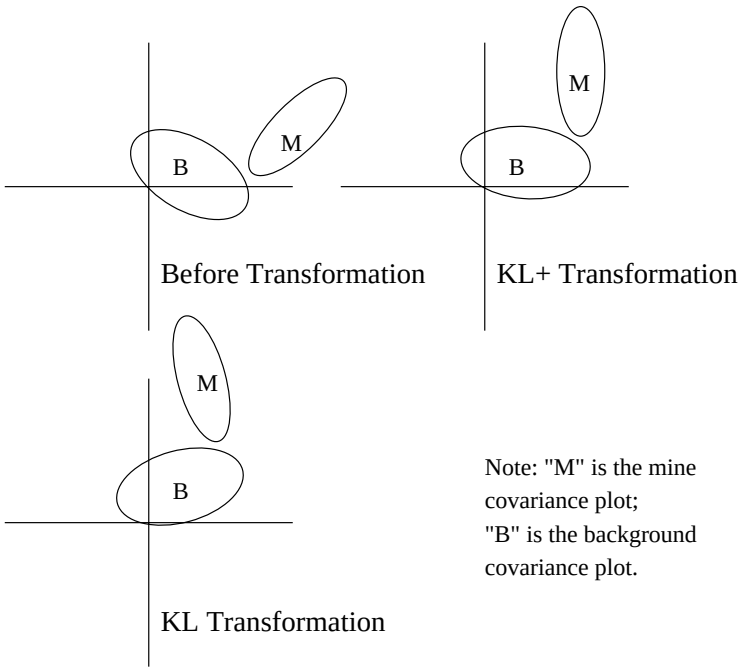


Figure 39: The KL+ Transform

both the A_{KL_+} transform matrix and a similar matrix computed from a covariance matrix of the background patterns. In Figure 40, this matrix is called the A_{KL_-} matrix.

The composite transform uses the most significant eigenvectors from both the A_{KL_+} and the A_{KL_-} matrices. To perform the composite transform, we form the matrix A_m as:

$$A_m = [A_{KL_+}[1 : m_1] : A_{KL_-}[1 : m_2]]$$

A_m is a $k \times m$ dimensional array which holds the first m_1 columns of A_{KL_+} and the first m_2 columns of A_{KL_-}. The values m_1 and m_2 are chosen as:

$$m_1 = \#\text{ of } \lambda_i \text{ from } A_{KL_+} \text{ greater than or equal to: } \overline{\lambda}_{KL_+}$$

$$m_2 = \#\text{ of } \lambda_i \text{ from } A_{KL_-} \text{ greater than or equal to: } \overline{\lambda}_{KL_-}$$

Once A_m is formed, the transform is carried out as outlined above:

$$Y_m = A_m^T X$$

This vector is used as input to either the NN or PNN, as appropriate, and training and testing is performed as usual.

7.4 The Eigenspace Separation Method

One of the transform methods investigated, the Eigenspace Separation Method (ESM), was developed by Dr. Torrieri of ARL. The following theoretical details and derivation are courtesy his personal communication [46].

begin {generate A_{KL_+}}

$N = 0$, $\phi = [0]_{k\times 1}$, $\Phi = [0]_{k\times k}$.

For each preprocessed vector X_i in the training section:

If X_i is not an outlier and X_i is a mine pattern, then:

$\phi \leftarrow \phi + X_i$

$\Phi \leftarrow \Phi + X_i X_i^T$

$N \leftarrow N + 1$

endif

endfor

$\Sigma_+ = \frac{1}{N-1}\left[\Phi - \frac{\phi\phi^T}{N}\right]$

Compute the eigenvectors u_i and the eigenvalues λ_i of Σ_+.

$A_{KL_+} = [u_1 u_2 \ldots u_k]$, sorted by descending λ_i.

$b_{KL_+} = [\lambda_1 \lambda_2 \ldots \lambda_k]^T$

Store $A_{KL_+}^T$, b_{KL_+} for later use.

end {generate A_{KL_+}}

begin {generate $A_{KL\_}$}

$N = 0$, $\phi = [0]_{k\times 1}$, $\Phi = [0]_{k\times k}$.

For each preprocessed vector X_i in the training section:

If X_i is not an outlier and X_i is a background pattern, then:

$\phi \leftarrow \phi + X_i$

$\Phi \leftarrow \Phi + X_i X_i^T$

$N \leftarrow N + 1$

endif

endfor

$\Sigma_- = \frac{1}{N-1}\left[\Phi - \frac{\phi\phi^T}{N}\right]$

Compute the eigenvectors u_i and the eigenvalues λ_i of Σ_-.

$A_{KL\_} = [u_1 u_2 \ldots u_k]$, sorted by descending λ_i.

$b_{KL\_} = [\lambda_1 \lambda_2 \ldots \lambda_k]^T$

Store $A_{KL\_}^T$, $b_{KL\_}$ for later use.

end {generate $A_{KL\_}$}

Figure 40: Algorithms for generating A_{KL_+} and A_{KL_-}

The purpose of the ESM is to perform a linear transformation which will maximize the difference in norm of patterns of class 1 and patterns of class 2. i.e. we wish to maximize the metric:

$$D = \left| E\left[\| y_1 \|^2 - \| y_2 \|^2 \right] \right|$$

.

We note that if X is an input vector, and A_m is the transformation matrix, then:

$$E[\| Y \|^2] = E\left[\sum_{j=1}^{m} (a_j^T X)^2\right] = E\left[\sum_{j=1}^{m} (a_j^T X)(X^T a_j)\right]$$

Computing the expected square norm for vectors X belonging to the mine class or the background class, we get:

$$E[\| Y_+ \|^2] = \sum_{j=1}^{m} a_j^T R_+ a_j \qquad E[\| Y_- \|^2] = \sum_{j=1}^{m} a_j^T R_- a_j$$

where R_+ is the correlation matrix for mine patterns and R_- is the correlation matrix for background patterns.

We define $M = [R_+] - [R_-]$. Therefore, we can write:

$$D = \left| \sum_{j=1}^{m} a_j^T M a_j \right|$$

Now, if we define:

$$E_p = \sum_{\substack{i=1 \\ \lambda_i > 0}}^{k} \lambda_i \qquad E_n = \sum_{\substack{i=1 \\ \lambda_i < 0}}^{k} \mid \lambda_i \mid$$

then, it can be shown that $D \leq \max[E_p, E_n]$. Therefore, we maximize D by choosing the transform A_m to include all of the eigenvectors corresponding to the positive eigenvalues of M if $E_p > E_n$, or by choosing the transform A_m to include all of the eigenvectors corresponding to the negative eigenvalues of M if $E_n > E_p$.

```
begin {generate A_ESM}
    N+ = 0, Φ+ = [0]k×k.
    N− = 0, Φ− = [0]k×k.
    For each preprocessed vector Xi in the training section:
        If Xi is not an outlier, then:
            If Xi is a mine pattern then:
                Φ+ ← Φ+ + Xi Xi^T
                N+ ← N+ + 1
            else
                Φ− ← Φ− + Xi Xi^T
                N− ← N− + 1
            endif
        endif
    endfor
    R+ = (1/N+) Φ+
    R− = (1/N−) Φ−
    M = [R+] − [R−]
    Compute the eigenvectors ui and the eigenvalues λi of M.
    A_ESM = [u1 u2 ... uk], sorted by descending λi.
    b_ESM = [λ1 λ2 ... λk]^T
    Store A_ESM^T, b_ESM for later use.
end {generate A_ESM}
```

Figure 41: Algorithm for generating A_{ESM}

The algorithm for generating the transform array A_{ESM} is shown in Figure 41. Two covariance matrix are maintained: one for mine patterns, and one for non-mine patterns. The matrix $M = [R_+] - [R_-]$ is computed and the eigenvectors of this matrix are used to form the transform matrix.

To perform the ESM transform, we create the matrix A_m from A_{ESM} as either:

$$A_m = A_{ESM}[1 : m], \text{ or } A_m = A_{ESM}[k - m + 1 : k].$$

Again, A_m is a $k \times m$ dimensional array which holds either the first or last m columns of A_{ESM}. We note that M is *not* a positive semi definite matrix (although it is still symmetric) since it is the difference of two other p.s.d. matrices. So, the eigenvalues of M may be negative. With this in mind, we choose m to be:

$$m = \max\{\#\text{ of positive } \lambda_i, \#\text{ of negative } \lambda_i\}$$

If the number of positive eigenvalues is greater than the number of negative eigenvalues, then $A_m = A_{ESM}[1:m]$; otherwise $A_m = A_{ESM}[k-m+1:k]$.

Once A_m is formed, the transform is carried out as outlined above:

$$Y_m = A_m^T X$$

This vector is used as input to either the NN or PNN, as appropriate, and training and testing is performed as usual.

7.5 The Simultaneous Diagonalization Transform

The purpose of the KL transform is to diagonalize the covariance matrix of a distribution by rotating the input space to align it with the axis. The KL transform has no knowledge of the various sub-covariance matrices which form the overall distribution, so conceivably knowledge of them could be used to simultaneously diagonalize two different covariance matrices, and thus improve on the KL transform. This method is called the "Simultaneous Diagonalization Transform".

First, let's look at it graphically in Figure 42. If we know the covariance matrix for the background patterns, we can compute the KL transform matrix $A_{KL\_}$ using that knowledge. Applying this transform to the input data aligns the background covariance matrix with the axes, but not the minetype covariance matrix. Now, let us whiten the $KL\_$ transform. Now, the background covariance matrix is the identity matrix, and it is aligned with any possible set of axes. So, thirdly, we can compute the transform of the rotated, scaled, minetype patterns, and transform the resulting data. Thus, the input set is rotated, scaled, and rotated again to form an overall covariance matrix whose sub-covariance matrices are diagonal[28] The derivation of the SDM algorithm is included in Appendix B.

The resulting transform is *not* orthonormal. Euclidean distances are not maintained. It is lossless, however, if all components of the transform array are used.

The criterion used to determine which "features" of A_{SDM} to retain and which to discard is rather complicated (It is covered in detail in Appendix C). Each vector in A_{SDM} is treated separately, and the MSE e_i^2 incurred if that row were removed is calculated. The array A_{SDM} is then sorted by decreasing e_i^2, and A_m is formed from A_{SDM} as:

[28] We also could have whitened with respect to Σ_+ and diagonalized with respect to Σ_-. However, we note that Σ_+ is in turn composed of $\Sigma_1, \Sigma_2, \ldots \Sigma_5$, the matrices of the 5 minetypes. The Σ_- matrix may be far more homogeneous, so whitening the transform with respect to it makes more sense.

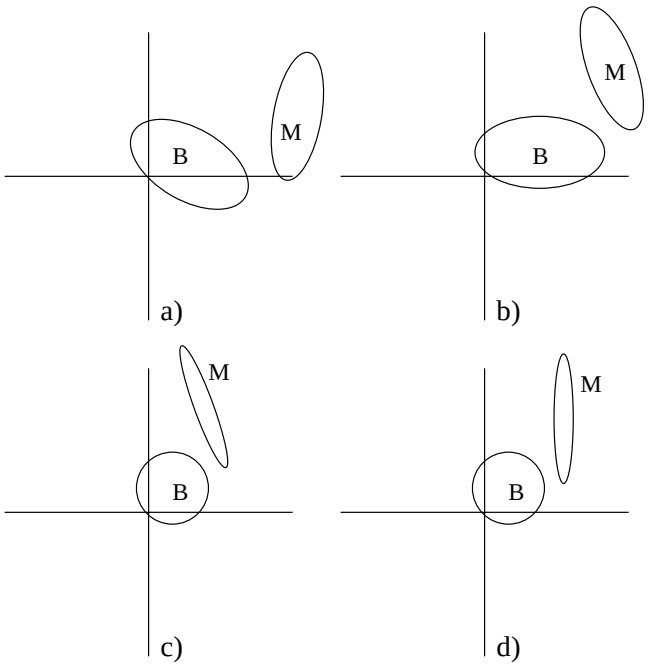


Figure 42: Formation of the Simultaneous Diagonalization Transform. a) The original covariance diagrams. b) Rotated w.r.t. the background patterns. c) Whitened w.r.t. the background patterns. d) Both matrices diagonalized.

$$A_m = A_{SDM}[1:m].$$

Again, A_m is a $k \times m$ dimensional array which holds the first m columns of A_{SDM}. The value of m is chosen as:

$$m = \min_j{}^{-1}\left[\frac{\sum_{i=1}^{j} e_i^2}{\sum_{i=1}^{k} e_i^2} \geq \frac{E}{100}\right], \quad 0 \leq E \leq 100$$

Once A_m is formed, the transform is carried out as outlined above:

$$Y_m = A_m^T X$$

This vector is used as input to either the NN or PNN, as appropriate, and training and testing is performed as usual.

7.6 The "UNKL" Transform

It was thought that there may be an advantage to re-mapping the transformed X back to the original higher dimensional space. i.e.[29] $Y = A_m[A_m^T X]$. Since $A_m^T X_i$ is a lossy operation, we are mapping X_i onto an m dimensional space and then re-mapping it back

[29]Note: Since the A_{SDM} array is not unitary, the UNKL operation for SDM is: $Y = A_m^{-T}[A_m^T X]$, where A_m^{-T} is the first m columns of $\Phi\theta^{\frac{1}{2}}\Psi$.

begin {generate A_{SDM}}

$N_+ = 0$, $\phi_+ = [0]_{k\times 1}$, $\Phi_+ = [0]_{k\times k}$.

$N_- = 0$, $\phi_- = [0]_{k\times 1}$, $\Phi_- = [0]_{k\times k}$.

For each preprocessed vector X_i in the training section:

If X_i is not an outlier, then:

If X_i is a mine pattern then:

$\phi_+ \leftarrow \phi_+ + X_i$

$\Phi_+ \leftarrow \Phi_+ + X_i X_i^T$

$N_+ \leftarrow N_+ + 1$

else

$\phi_- \leftarrow \phi_- + X_i$

$\Phi_- \leftarrow \Phi_- + X_i X_i^T$

$N_- \leftarrow N_- + 1$

endif

endif

endfor

$\Sigma_+ = \frac{1}{N_+ - 1}\left[\Phi_+ - \frac{\phi_+ \phi_+^T}{N_+}\right]$

$\Sigma_- = \frac{1}{N_- - 1}\left[\Phi_- - \frac{\phi_- \phi_-^T}{N_-}\right]$

Compute the eigenvectors Φ and the eigenvalues θ of Σ_-.

Set $R = \theta^{-\frac{1}{2}} \Phi^T \Sigma_+ \Phi \theta^{-\frac{1}{2}}$

Compute the eigenvectors Ψ and the eigenvalues Λ of R.

Set $A_{SDM}^T = \Psi^T \theta^{-\frac{1}{2}} \Phi^T$

Compute the individual MSE e_i^2 for removing row i from A_{SDM}.

Sort A_{SDM} by decreasing e_i^2.

$b_{SDM} = \left[e_1^2 e_2^2 \ldots e_k^2\right]^T$

Store A_{SDM}^T, b_{SDM} for later use.

end {generate A_{SDM}}

Figure 43: Algorithm for generating A_{SDM}

to the original k dimensional space. The network is given more coefficients to work with, which may aid it in finding a good decision boundary (through a sparser space). Another way to look at it is that the most significant noise was eliminated in $(A_m^T X_i)$, and that the reprojection puts the vector back into the higher dimensional space with the noise removed.

8 Vector Quantizer Aided Generalization

The ability of a neural network to generalize forms the basis of using them for the mine detection problem. Clearly, there will always be a practical limit to the amount of data that can be collected to train the network. At some time, the network must be able to process new data and decide whether or not a mine is present at that point. In both the PNN and NN, but most explicitly with a PNN, this generalization is an intelligent interpolation in k-dimensional space between the known data points.

Figure 44 shows two interpolating curves that pass exactly through a discrete set of data points. With no further knowledge of the underlying function that is being approximated, either of these two curves may be the better fit to the true data. In fact, there are an infinite number of curves which pass exactly through the data points, only one of which represents the true function.

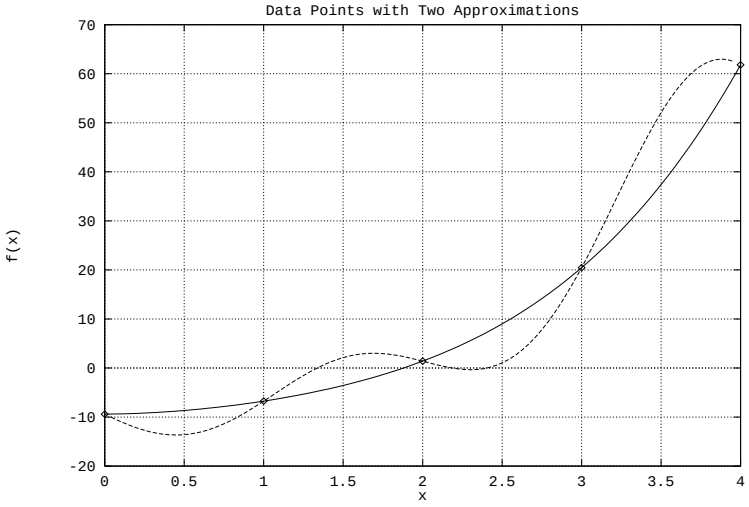


Figure 44: Two interpolating curves for a small data set.

If, furthermore, we assume that the data points may be noisy measurements of some true data points, then the "correct" curve may not pass exactly through any of the data points at all, but may look more like a curve obtained using the least-squares data fitting method.

The problem of pattern recognition/classification is exactly the same problem as fitting a function to a set of data points. It is a dual problem – it can be viewed as generating a decision boundary curve from data points, or it can be viewed as generating probability density surfaces, from which a decision boundary may be computed. In either case we are faced with a problem. Which curve best represents the underlying function? It is a widely held belief that the "best" curve is the one which most smoothly fits the data with the fewest oscillations. This conjecture is based on a belief in the simplicity and elegance of nature, and also on an assumption of the uniformity of noise.

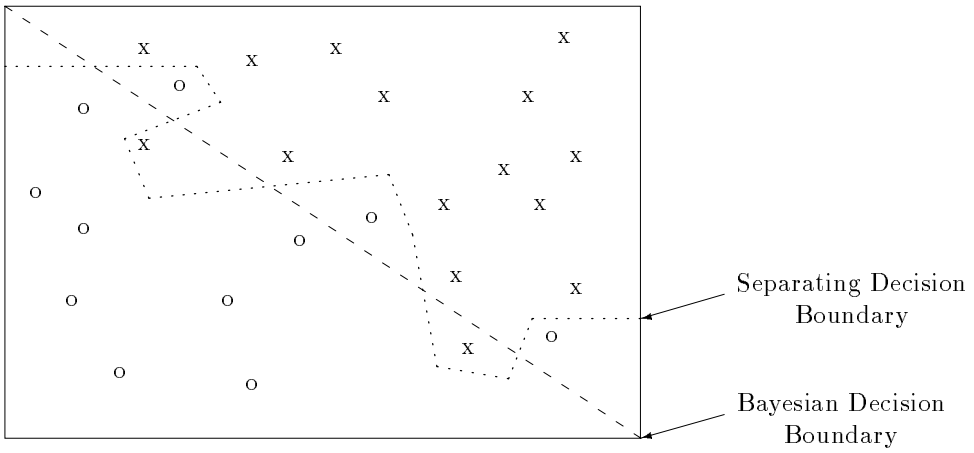


Figure 45: Two boundary curves.

Figure 45 shows an illustrative two-class pattern classification problem. The training data for class 1 is represented by the letter "o" and the training data for class 2 is represented by the letter "x". These data points represent noisy samples from two underlying probability density functions. Suppose that we know that the two classes are disjoint, that true "o" samples only occur in the lower triangular region, and that true "x" samples only occur in the upper triangular region. Then, the Bayesian decision boundary is the line separating these regions (assuming uniform noise). However, using the training samples shown, a network trying to minimize error over the training set might be trained to produce the separating line that is shown in the figure. As can be seen, this line perfectly separates the training samples; however, it will perform poorly on testing samples drawn from the underlying probability distribution functions. One method to solve this problem is to obtain more training samples to train the network with. Given a sufficiently large set of training samples, the network can learn the correct decision boundary. However, obtaining such a set of data is not always possible. For the case of mine detection, for example, we have a fixed data set with which to work.

One way to help remedy this problem is to discard or otherwise suppress the statistically unlikely points when training the network. The hope is that a more natural boundary will be formed. For this two-dimensional, two-class problem, we may easily visualize a decision boundary that is close to the Bayesian boundary. However, for the mine classification problem there are potentially six output categories and 88 dimensions to each input vector, which makes the statistical anomalies harder to detect. Several methods were tried, and are described in the following sections.

Since it was just stated that increasing the training set size will help the network learn the decision boundaries more correctly, why then should we expect the network to do better by decreasing the training set size (which we must do if we are to suppress the statistically unlikely points)? On one hand, many training points will help the network to

learn the boundary. On the other hand, the reason we need so many points is that some of them are statistically unlikely and must be compensated for. If they are removed, the network should be able to learn without so many training samples.

8.1 Data Clustering via Vector Quantization

This section describes several statistically motivated techniques to remove or de-emphasize statistical outliers. All methods use vector quantizer (VQ) algorithms, which are more commonly associated with the field of "lossy" data compression.

When vector quantization is used as a data compression technique, it works as follows. First, we assume that we have a "codebook" of vectors called "centroids",

$$\{Y_k : k = 1, 2, \ldots, r\}$$

each having the same dimension as a vector in our input data source. Then, for every input vector X from our input data source, we find its nearest neighbor in the codebook. This is done by finding the index i such that some distortion function $D(Y_i, X)$ is minimized. Typically, $D(a, b) = \|a - b\|^2$. Finally, the index i replaces the vector X. Since r is relatively small, the integer value i will take fewer bits to represent than the real valued vector X, and data compression is achieved. The decoder reverses this process to produce $\hat{X} = Y_i$.[30]

In order to optimize the compression, so that the reconstruction is as close as possible to the original, the codebook used for compression must be optimized. There are several ways to generate such a codebook; the most popular method is referred to as either the Lloyd I algorithm, or the LBG algorithm. An alternate method by Kohonen is called Learning Vector Quantization (LVQ). Both methods are iterative ways to develop an optimal codebook using a training set. Once the codebook is generated using either method, its contents (termed *centroids*) represent "typical" vectors in the training set.

VQ algorithms will be covered later. For now, we assume that a procedure exists to produce (near) optimal codebooks.

8.1.1 Statistical Outlier Suppression using VQ

We have seen how vector quantizer techniques can be used to compress an information source, but the question remains "How can they be used with networks for pattern recognition?" There are two different ways that a vector quantizer codebook can be used to help the generalization of a network, and the utility of each method will depend on the data itself.

The first method is to suppress the outliers. This is done by first developing a codebook of centroids. Typically a 10:1 (or even higher) ratio of training points to codebook vectors is desirable. Then, each of the centroids will be an average of approximately ten nearest neighbor input vectors. These centroids are then used to train a neural network.

[30] We note that $\hat{X} \neq X$, and so this data compression technique is "lossy".

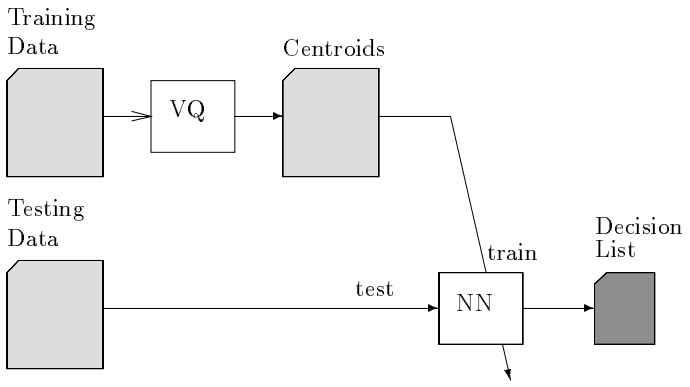


Figure 46: Data flow when using a VQ to suppress outliers.

That is to say, the neural network is no longer trained using data directly taken from the mine lane, but is trained with the centroids computed by the VQ algorithm. This is shown in Figure 46.

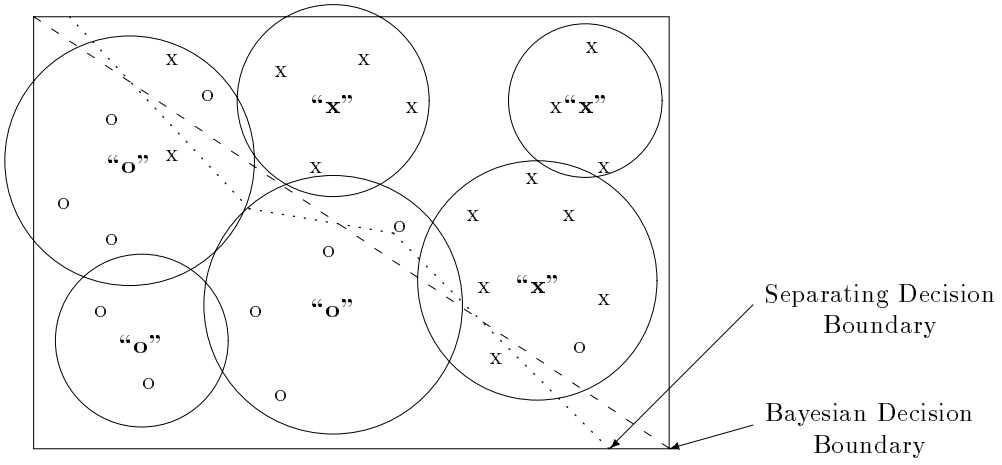


Figure 47: Boundary curve trained using centroids.

Figure 47 illustrates how this process works. The large circles on the figure show the rough areas covered by each centroid. The centroids themselves are depicted as boldface "x" or "o". We see that each centroid is on the correct side of the Bayesian boundary, and thus the separating decision curve is much closer to the optimal one.

By training with the centroids, the network is learning a smoother function since noise has been suppressed. If noise is a problem, generalization should improve.

8.1.2 Statistical Outlier Removal using VQ

Thus far, we have not described how desired responses are associated with centroids. This is discussed in Section 8.3. We simply note for now that it is possible for the Voroni

region defined by a certain centroid to contain input vectors with several different desired responses. These are input vectors which are close to the Bayesian decision boundary.

In this case we may assume that those input vectors whose class does not agree with the class of their nearest centroid are statistical anomalies. The "statistical outlier removal" method simply deletes these input vectors from the training set.

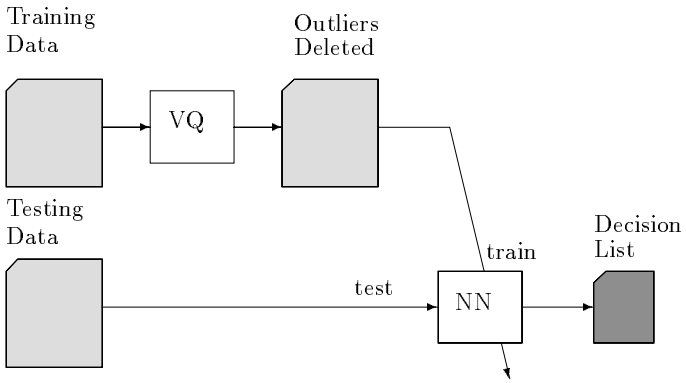


Figure 48: Data flow when using a VQ to remove outliers.

Figure 48 shows how this is done. The process is very similar to that used when suppressing outliers. The VQ algorithm is once again run on the training data. This time, the centroids are not used to train the network. Instead, the outliers are thrown away and the remaining training vectors are used to train the neural network.

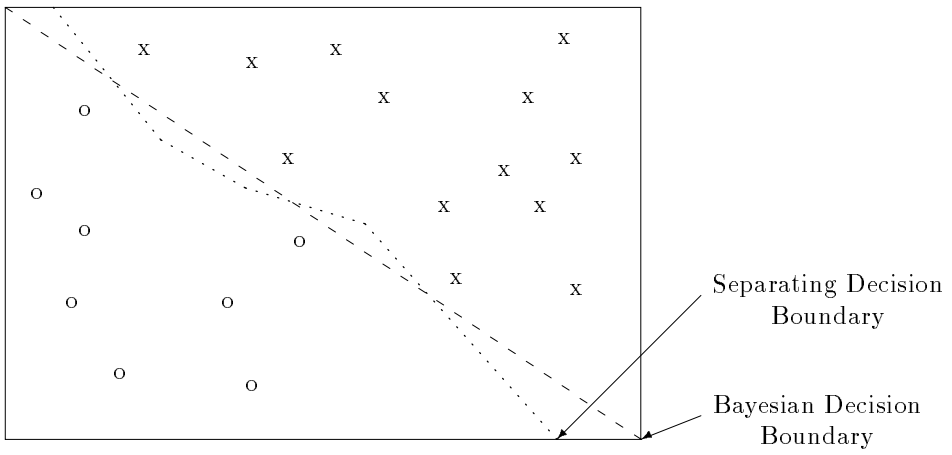


Figure 49: Boundary curve trained without statistical outliers.

Figure 49 illustrates how this process works. With the outliers removed, the "x" and "o" points are all on the correct side of the Bayesian boundary, and thus the separating decision curve is much closer to the optimal one.

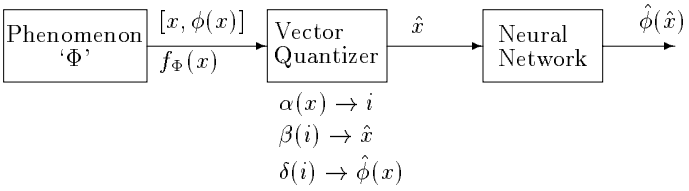


Figure 50: Data flow in a pattern recognition system incorporating a vector quantizer

8.2 Vector Quantizer Methods

We now expand a bit on the theory of VQ, the different algorithms used to construct codebooks, and the rationale behind, and subtle side-effects of each. Material for this summary was obtained from a wide variety of sources [10, 28, 29, 25, 34, 51, 36, 47, 2]. Since each original researcher has their own notation and style, it was necessary to develop a unified notation to compare algorithms from these different sources. This notation agrees with the original sources as well as possible without conflicting with itself.

The philosophy behind vector quantization is to represent some given phenomenon as accurately as possible (in some sense) with finite resources. Since the phenomenon to be represented can not typically be defined by a finite set of descriptors, there will be a certain degree of inaccuracy in the reconstruction, termed "distortion." A description is usually considered optimum if it minimizes this distortion[31].

Let us consider only those phenomena $\Phi \subseteq \Re^k$: those Φ which are a subset of the class of all real valued vectors of dimension k. Furthermore, let us consider that examples, 'x', from this phenomenon have a class function (not necessarily known) $\phi(x)$, and a probability density function (pdf): [32]

$$f_\Phi(x) = \begin{cases} 0 \leq f_\Phi(x) \leq 1 & x \in \Phi \\ 0 & x \notin \Phi \end{cases}$$

$$\text{such that } \int_\Phi f_\Phi(x) = 1$$

Therefore, examples of this phenomenon will be ordered pairs of the form $[x, \phi(x)]$, and will occur with frequencies proportional to $f_\Phi(x)$. This is illustrated on the left hand side of Figure 50.

A fully trained vector quantizer (VQ) is shown in the center of Figure 50. The training of the VQ consists of constructing the functions $\alpha(x)$, $\beta(i)$ and $\delta(i)$. $\alpha(x)$ is the "encoding function" which converts the input example to the quantized output 'i'. The

[31] Note that in Sections 8.4.4 and 8.4.4 the desired solutions do not minimize distortion. Therefore, we see that other cost functions can be used to grade the representation.

[32] The mine lane data clearly belongs in Φ: at each spatial point in the lane, we measure a vector of 88 real valued coefficients. The class at each point takes on the value '0' if the point is a background point, or the values '1' to '5' if the point is over a mine. The pdf is unknown, and can only be estimated from the data.

"decoding function" $\beta(i)$ in turn creates an estimate of the input, '$\hat{x}$', from this quantized value. $\delta(i)$ is not always used, but its function is to estimate the class of the input, $\hat{\phi}(x)$, given the quantized value. When used with a neural network to perform pattern recognition, the VQ outputs $\hat{x} = \beta(\alpha(x))$ which the neural network uses as input to produce the class estimate $\hat{\phi}(\hat{x})$. A VQ distortion function $D(\hat{x}, x)$ is defined to quantify the quantization error between $\hat{x}$ and x. For our work, a simple mean-square-error distortion measure is used, so the total distortion function to be minimized is $D(\hat{x}, x) = \|\hat{x} - x\|^2$. $\alpha(\cdot)$ and $\beta(\cdot)$ are chosen to minimize the expected distortion $E_x\left[D(\beta(\alpha(x)), x)\right]$.

The actual functions $\alpha(\cdot)$ and $\beta(\cdot)$ are realized in the vector quantizer by constructing a "codebook" of "centroids". The term 'codebook' simply defines a set '$\mathcal{C}$':

$$\mathcal{C} = \{Y_k : k = 1, 2, \ldots, r\},$$

where each Y_k is a vector of the same dimension as x, and is called a 'centroid.' The function $\alpha(x)$ returns the index i of the centroid which represents x with the lowest distortion:

$$i = \alpha(x) = \min_k{}^{-1}\left[D(Y_k, x)\right].$$

The function $\beta(i)$ returns the centroid vector indexed by i: $\beta(i) = Y_i$. The $\delta(\cdot)$ function is realized by constructing an array of classes, c_k which correspond to Y_k. Then, $\delta(i) = c_i$.

While this overall concept of vector quantization is fairly uniform in the field, there are a number of different algorithms available for designing VQ codebooks. None of these algorithms guarantees even that the "optimal solution" is met; however, they will all provide "good" solutions (locally optimal ones) in terms of the error function they are trying to minimize. The algorithms typically differ in one of three categories: batch versus stochastic update learning; difference in the error function being minimized; and supervised versus unsupervised learning. Understanding how the algorithms differ in these ways can be a key to understanding the fundamental differences between results obtained with them. The particular algorithms which will be addressed here are: LBG (Lloyd-I), LVQ 1, LVQ 2.1, LVQ 3, OLVQ1, SOM, "SOM with a conscience", FSCL and C&SL.

8.2.1 Optimality Conditions

There are three known necessary (but not sufficient) conditions for a vector quantizer to be optimal in the sense of minimizing distortion. They are called: the "Nearest Neighbor Condition", the "Centroid Condition" and the "Zero Probability Boundary Condition". The only algorithm which guarantees that these optimality conditions are met is the LBG algorithm. However, even the LBG algorithm may produce codebooks which are not globally optimal. The conditions will be described, and then a very simple counter-example will be given to show that a globally optimal codebook is not always achieved.

Optimality: Nearest Neighbor Condition

The first optimality condition, the "Nearest Neighbor Condition", optimizes the encoder for a specific decoder. It states that every input data point must be assigned to the centroid which minimizes the distortion function:

$$\alpha(x) = i = \min_k {}^{-1} [D(Y_k, x)].$$

This is clearly a necessary condition since, if an input data point were assigned to some other centroid, the total distortion would be decreased by simply remapping it to its nearest neighbor. The nearest neighbor condition partitions the input data source into regions R_k (called "Voroni Regions") represented by each centroid Y_k. The union of all R_k is the entire input set R: $R = \bigcup_k R_k$.

Optimality: Centroid Condition

The second condition, called the "Centroid Condition" optimizes the decoder for a specific encoder. It states that every centroid must be equal to:

$$\beta(i) = Y_i = \text{cent}(R_i) = \min_y {}^{-1} E[D(y, X)|X \in R_i]$$

That is, for every Voroni region R, its centroid minimizes the total expected distortion between itself and all points in that region. If the distortion function is the squared error function, this reduces to: (for a discrete source)

$$\text{cent}(R_i) = \frac{1}{\|R_i\|} \sum_{k=1}^{\|R_i\|} x_k$$

Optimality: Zero Probability Boundary Condition

The third optimality condition states that an optimal codebook will meet the condition:

$$P\left(\bigcup_{k=1}^{r} B_k\right) = 0,$$

that is, examples 'x' falling on the boundary between two Voroni regions occur with zero probability. We see that this must be the case by considering an input point x_0 which is encoded to Y_j but is equally distant to Y_i. If we move x_0 to the region R_i, then a different code is made with the same average distortion. This results, however, in moving a nonzero probability input point from R_j to R_i, which must move the centroids of both R_j and R_i, which means that the codebook is no longer optimal for the new partition.

This condition is clearly degenerate when the input is a continuous random variable, since all boundary points will have zero probability. However, when the input is a discrete random variable, this condition has meaning.

A Counter Example

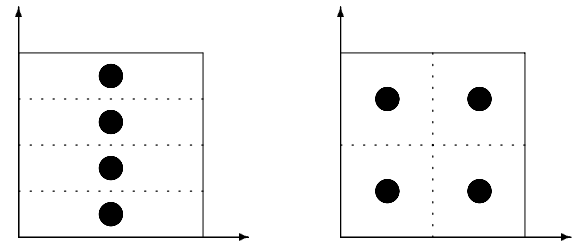


Figure 51: Two locally optimal codebooks

If *all three* of these conditions are simultaneously met, then the codebook of centroids is locally optimized. However, the codebook may not be globally optimized. Consider for instance, an input with x_1 and x_2 both uniformly distributed between 0 and 1. Suppose we have a codebook of four centroids $\mathcal{C}_1$ which are positioned at

$$\mathcal{C}_1 = \{(0.5, 0.125), (0.5, 0.375), (0.5, 0.625), (0.5, 0.875)\}.$$

These centroids are locally optimal, and satisfy the three optimality conditions. However, the codebook $\mathcal{C}_2$ of four centroids:

$$\mathcal{C}_2 = \{(0.25, 0.25), (0.25, 0.75), (0.75, 0.25), (0.75, 0.75)\}$$

produce a codebook with lower distortion. See Figure 51 for an illustration of this example. The ability of LBG, or any of the other methods (which are all descent methods), to find the globally optimal solution depends entirely on the initial codebook used. Typically, local solutions are still quite good, but methods exist to help select the initial codebook to enable better results. These are described in Section 8.4.5.

8.3 Bayes Risk Weighted VQ

When using Vector Quantizers to design a codebook of centroid vectors for pattern recognition purposes, it is being simplistic to assume that a simple squared-error distortion cost function will produce the optimum codebook. The cost function should really include one other element: the cost associated with misclassifying a point [34, 51, 36].

To incorporate this "Bayes risk" term into the VQ design, we need to redefine our cost function J_λ as follows:

$$\begin{aligned} J_\lambda(\alpha,\beta,\delta) &= D(\alpha,\beta) + \lambda B(\alpha,\delta) \\ D(\alpha,\beta) &= E\left[\|X - \beta(\alpha(X))\|^2\right] \\ B(\alpha,\delta) &= \sum_{k=0}^{M-1}\sum_{j=0}^{M-1} C_{jk} Pr(\delta(\alpha(X)) = k \text{ and } Y = j) \end{aligned}$$

$D(\cdot)$ is the mean squared error distortion term we have used before.

$B(\cdot)$ is the Bayes risk term.

$\alpha(\cdot)$ is the VQ encoding function: $i = \alpha(X)$.

$\beta(\cdot)$ is the VQ decoding function: $\hat{X} = \beta(i)$.

$\delta(\cdot)$ is the mapping between codeword indices and output classes. A classification error occurs when $\delta(\alpha(X)) \neq Y$, where $\{X, Y\}$ are the corresponding input data points and desired responses.

C_{jk} are the costs of classification errors when a vector of class j is misclassified as class k. The C_{jk} matrix used for all experimental work here is tabulated in Table 1.

λ weighs the cost function between pure distortion when $\lambda = 0$ and pure Bayes risk when $\lambda = \infty$.

		Class Estimate					
		Bgnd	m_1	m_2	m_3	m_4	m_5
True Class	Bgnd	0	1	1	1	1	1
	m_1	2	0	0.1	0.1	0.1	0.1
	m_2	2	0.1	0	0.1	0.1	0.1
	m_3	2	0.1	0.1	0	0.1	0.1
	m_4	2	0.1	0.1	0.1	0	0.1
	m_5	2	0.1	0.1	0.1	0.1	0

Table 1: C_{jk} classification costs.

This new Bayes risk term forces the VQ design program to generate a set of centroids which take into account the costs associated with misclassification errors. The relative costs of these errors used for this research are tabulated in Table 1. We see that the cost associated with classifying a point correctly is zero. The cost associated with misclassifying one mine-type as another mine-type is a minimal constant. Since we are not too concerned (at this point in time) with classifying mine-types individually, the cost is not large. Neither is it zero, however, as this could introduce confusion in the VQ centroid generation process. The cost associated with misclassifying background as mine is 1 and the cost associated with misclassifying a mine as background is 2 to indicate that this is a more serious error.

In this enhanced cost function J_λ, the variable λ weighs the two terms. When $\lambda = 0$, the cost function is completely distortion based, and when $\lambda = \infty$, the cost function

is completely Bayes risk based. In fact, personal correspondence with Wesel, (one of the authors of [51]) has discovered that the best results are always obtained in practice when λ is either zero or infinity. Reducing the problem to these two cases simplifies the process of designing a VQ considerably.

8.3.1 Method 1: $\lambda = 0$

For the $\lambda = 0$ case, we consider only the MSE distortion when designing a codebook. For this, we use the standard VQ design algorithm. A set of r locally optimal centroids are generated in an unsupervised manner. Then, desired responses are generated for each centroid. This is done by counting the number of vectors of each class in every cluster (n_i) and computing the misclassification cost associated with it. The desired response for that centroid is the class j which minimizes the misclassification cost:

$$\text{Desired Response} = \min_j{}^{-1} \sum_{\text{classes } i} n_i C_{ij}$$

8.3.2 Method 2: $\lambda = \infty$

For the $\lambda = \infty$ case, the cost function is based entirely on Bayes risk. Therefore, we force centroids to be trained only on vectors from their own class. The procedure for generating these centroids then turns out to be the same as splitting the input data into six sets according to desired response, and individually generating centroids for each set. Labeling the desired responses for the centroids is simple since they were only trained using data from one class – the desired response is simply that class.

8.3.3 Method 3: $\lambda = \infty$, reclassify

One subtle detail arises when using method 2. Notice that the cost function $B(\cdot)$ is a function of the encoding function $\alpha(\cdot)$ *and* the mapping function $\delta(\cdot)$. The procedure described above optimizes the $\alpha(\cdot)$ function, but completely ignores the $\delta(\cdot)$ function. It turns out that a much better VQ classifier can be built if $\alpha(\cdot)$, $\beta(\cdot)$ and $\delta(\cdot)$ are all optimized together [36]. Unfortunately, the NN or PNN structure does not afford us this opportunity. They both implicitly use a $\delta(\cdot)$ function that resembles a k-nearest neighbor mapping. For such systems, it has been found that improved results can be obtained by designing a $\lambda = \infty$ set of centroids, and then reclassifying them [51]. This additional reclassification step keeps the centroids generated by Method 2, but reassigns classes j to each centroid according to:

$$\text{Desired Response} = \min_j{}^{-1} \sum_{\text{classes } i} n_i C_{ij}$$

8.3.4 Some Visual Examples

For illustrative purposes, two examples are considered. The first example is for a completely disjoint two-class problem. The two probability distribution functions are uniform. This is

shown in Figure 52 as two grey boxes. Either of the three VQ design methods may be used to generate good centroids for this case. Methods 2 and 3 will produce identical results since there would be no misclassified points for any centroid. If the centroids initially assigned randomly at the start of method 1 are evenly split between the two distributions, then it will produce results identical with methods 2 and 3 as well. Otherwise, it will end up with more centroids in one distribution than the other, and performance will be poorer.

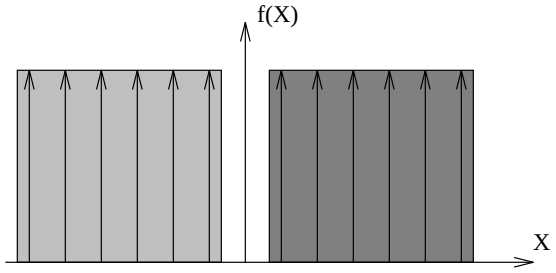


Figure 52: Two Disjoint Distributions.

For the second illustrative example, consider the two probability distribution functions in Figure 53. Again, both are uniform, but they overlap each other. One distribution has width 2A and height 1/2A, and the other has width 4A and height 1/4A. The Bayesian decision boundary in this case (with equal *a priori* probabilities and loss functions) is at $\pm$A. Points inside this region should be associated with the dark pdf, and points outside should be associated with the light pdf. We note that there will be high residual error in this case.

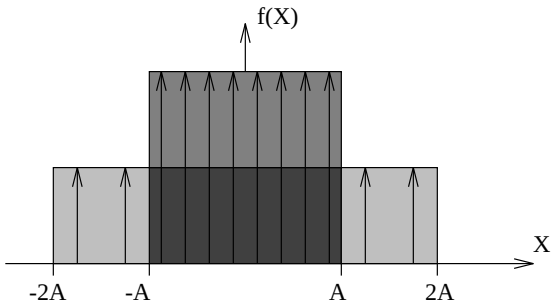


Figure 53: Overlapping Distributions: Method 1.

The $\lambda = 0$ method will assign 3/4 of its centroids to the center region since it has 3 times the probability of occurring as the outside regions. All of these centroids will be classified correctly according to the Bayesian criteria. The $\lambda = \infty$ case, however, will do very poorly. It will assign equal numbers of centroids to each pdf, and distribute them evenly. Figure 54 shows that half of the centroids in the center region are assigned to the light pdf, resulting in many non-Bayesian classifications. Such a system would perform classification much like the one in Figure 45. It may work well on the training data, but will perform poorly on testing data.

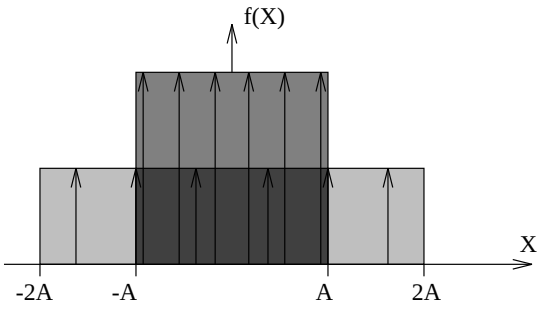


Figure 54: Overlapping Distributions: Method 2.

We see from Figure 55 that the $\lambda = \infty$ case with reclassification will perform much better on this example. All vectors are classified correctly, but the distribution is still not as good as in Figure 53.

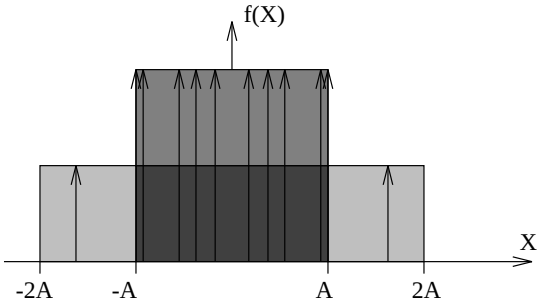


Figure 55: Overlapping Distributions: Method 3.

These examples were intended to show that either methods 1 or 3 are expected to perform best on the data. Which one is actually best will depend on the pdf of the data. Method 3 will work best on disjoint data, and method 1 will perform best on overlapping data.

8.4 VQ Design Algorithms

8.4.1 LBG: The Unsupervised/Supervised Batch Algorithm

All of the VQ design algorithms are descent algorithms – that is, they iteratively modify the centroid positions to cause the system error to move down some error curve. The LBG algorithm is distinguished from the other algorithms by being a "batch" algorithm. That is, it bases each centroid update on all of the input data. The other algorithms base a centroid update on a single example from the input source.

The LBG algorithm has the most theoretical background. It was originally invented by Lloyd[33]. It is the only one which guarantees that the optimality conditions are met, and

[33]Lloyd invented two algorithms for scalar quantizers, which he labeled 'I' and 'II'. The Lloyd-I algorithm is the one which was generalized to vector quantizers by Linde, Buzo and Gray. The VQ design algorithm

```
begin {LBG Algorithm}

    initialize centroids
    repeat
        repeat
            calculate distortion between input and centroids:
                d_1 = Σ_{i=1}^{‖R‖} D(β(α(x_i)), x_i).
            update the centroids with a Lloyd iteration
            check for empty cells
            calculate distortion between input and centroids:
                d_2 = Σ_{i=1}^{‖R‖} D(β(α(x_i)), x_i).
        until |d_1 − d_2| < ε
        check for boundary points and reassign them
    until no boundary points

end {LBG Algorithm}
```

Figure 56: The LBG Algorithm

it does so by construction. The algorithm is listed in Figure 56.

The initialization of the centroids is an important consideration for all of the algorithms, but is most important for the LBG algorithm. The LBG algorithm, being a batch update algorithm has monotonically nonincreasing error into the local minimum closest to the initial point. The other, stochastic, algorithms may be able to jump out of local minima and find a better solution, making the choice of initial conditions not so critical. Since the initialization of centroids is common to all the algorithms, it is discussed in Section 8.4.5.

begin {Lloyd Iteration number m}

1) Given a codebook $\mathcal{C}_m = \{Y_i\}$, partition the training set into cluster sets R_i using the Nearest Neighbor condition:

$$R_i = \{x \in R : D(Y_i, x) < D(Y_j, x);\ \text{all } j \neq i\}$$

For all ties, $D(Y_i, x) = D(Y_j, x)$, assign x to the region with lowest index 'i'.

2) Using the centroid condition, compute the new centroids for the cluster sets just found to obtain an new codebook $\mathcal{C}_{m+1} = \{\text{cent}(R_i)\}$.

end {Lloyd Iteration number m}

Figure 57: A Lloyd Iteration

A Lloyd iteration has two steps, which are outlined in Figure 57. If an empty cell j is generated in Step 1, then the largest cell (the one with max $\|R_i\|$) is split by assigning

bears the initials of the latter three researchers.

half of its elements (arbitrarily) to cell j[34].

The vector quantizer can operate in either a supervised or unsupervised manner[35]. To train in an unsupervised manner, simply run the LBG algorithm using all the input data for the r centroids you wish to produce. To train in a supervised manner, first divide the r centroids into r_i centroids for each class: $r = \sum_{\text{classes } i} r_i$. Then, for each class, train its r_i centroids using the LBG algorithm with only the input data of that class. In either case, the desired response for the centroid m, c_m is:

$$c_m = \min_j {}^{-1} \sum_{\text{classes } i} n_i C_{ij},$$

where, n_i is the number of input data points of type i in region R_m.

C_{ij} is the cost matrix for misclassifying an input point of type i to type j. The cost matrix used for all work to date is tabulated in Table 1.

Lastly, the LBG algorithm attempts to minimize total distortion; that is, to optimally fit the underlying probability distribution function of the input source. An alternate approach is discussed in Section 8.4.6, where the two methods are compared and contrasted.

8.4.2 SOM: The Unsupervised Stochastic Algorithms

The Self-Organizing Maps define a mapping from the input data space $\Re^k$ onto a regular two-dimensional array of nodes. Each node has a spatial position defined on a lattice, and the lattice can be arranged in either a rectangular or hexagonal pattern. A centroid vector Y_k is associated with every node k. SOM is a stochastic algorithm: centroid updates are based on a single input sample $x(n)$ at time index n. As with the LBG algorithm, an input vector $x(n)$ is compared with the Y_k, and the best match (usually in terms of MSE distortion) is defined as the "response": the input is thus mapped onto this 2D location[36].

$$i = \alpha(x(n)) = \min_k {}^{-1} [D(Y_k, x(n))].$$

In case of a tie, chose the i with smallest index. Thus, $x(n)$ is mapped onto the node i according to the centroid values Y_k.

An "optimal" mapping would be one that maps the probability density function $f_\Phi(x)$ in the most "faithful" fashion, trying to preserve at least the local structure of $f_\Phi(x)$. As such, those nodes in the array of centroids which are topographically close will be

[34] Alternately, the cell with the highest partial distortion could be split.

[35] This was discussed previously under the guise of "Bayes Risk Weighted VQ", where the unsupervised learning occurred with $\lambda = 0$ and the supervised learning occurred with $\lambda = \infty$ with reclassification. It is theoretically uncertain which of these approaches leads to the best solution – it is data dependent. Supervised learning seems to work best for the mine lane data.

[36] One might say that the SOM is a "nonlinear projection" of the probability density function of the high-dimensional input data space onto the two dimensional map space.

from input vectors which are close together in the input space. This is enforced by the neighborhood concept of the SOM learning algorithm.

The initial values of the codebook can be random or even entirely zero. For every input example $x(n)$, we find the response of the network $i = \alpha(x(n))$, and update all centroids as follows:

$$Y_k(n+1) = Y_k(n) + h_{ik}(n)[x(n) - Y_k(n)],$$

where h_{ik} is the so-called neighborhood kernel; it is a function defined over all lattice points. Usually $h_{ik}(n) = h(\|r_i - r_k\|, t)$, where $r_i, r_k \in \Re^2$ are the position vectors of nodes i and k in the array. With increasing $\|r_i - r_k\|, h_{ik} \to 0$. The average width and form of h_{ik} defines the "stiffness" of the elastic surface to be fitted to the data points.

Two widely used neighborhood kernels are the "bubble" kernel and the "Gaussian" kernel. For the bubble kernel, $h_{ik}(n) = \xi(n)$ if $\|r_i - r_k\| < \rho(n)$ and 0 otherwise, where $\xi(n)$ and $\rho(n)$ are monotonically decreasing functions. A "bubble" of nodes within radius $\rho(n)$ surrounding the response node are all updated with the same strength. As the radius decreases, the stiffness of the elastic surface decreases, and updates are more local in nature. As the ξ function decreases, smaller and smaller changes are allowed, so that the mapping converges and does not become unstable.

For the Gaussian kernel,

$$h_{ik}(n) = \xi(n) \exp\left(-\frac{\|r_i - r_k\|^2}{2\sigma^2(n)}\right),$$

where $\xi(n)$ is a learning rate as before, and $\sigma(n)$ defines the width of the kernel. $\sigma(n)$, $\rho(n)$ and $\xi(n)$ are usually linearly decreasing functions.

As with all other methods, the initial values of the centroids will affect the final centroid values. Due to the geometric structure of the centroid array using SOM, the normal centroid initialization procedures may not work well, so a random initialization could be best. In this case, however, it is a good idea to run the algorithm several times and choose the set of centroids which represents the input phenomenon with the lowest distortion.

The SOM algorithm is an unsupervised learning algorithm. It is also a stochastic algorithm – centroid updates are based on a single example $x(n)$.

8.4.3 SOM With a Conscience

One of the inherent problems with the SOM algorithm is that it is uncertain how to initialize the centroids properly. For all the other methods, there exist algorithms which create good initial conditions. But, due to the neighborhood kernel requiring spatially close (on the 2D centroid plane) centroids to be close in the multi-dimensional space too, the problem of initialization becomes non-trivial. If this did not affect the results, we would not care, but unfortunately, it can cause several damaging side effects.

At the termination of the SOM learning algorithm, we would like centroids which are spread out to divide the input space into discrete, equiprobable regions[37]. Consider a simple example where the initial distribution is poor and SOM fails: Make the input probability density function uniform over the real numbers in the interval [1..2]. Initialize a set of r centroids to uniformly cover the interval [0..3]. Recall that the SOM response is the centroid nearest the input point. This will always be a centroid within, or the first closest to, the interval [1..2]. Any centroids outside this region are never chosen as the SOM response. With a neighborhood kernel equal to $\delta(x)$, we see that these centroids will never be updated, and at the end of the algorithm they will still lie outside of the interval [1..2][38]. Since the position of the centroid is supposed to represent probability mass, and centroids exist where there is no probability of an input occurring, SOM has clearly failed.

An algorithm has been developed which solves this initialization problem, and even speeds up SOM learning by roughly an order of magnitude. Rather than directly optimizing the initial centroids, "SOM with a conscience" (which we will call SOMC) changes the learning algorithm to make sure that all centroids will be updated at roughly the optimum rate.

Since the desired end product will result with centroids which define equiprobable regions, at the end of training each centroid will be updated about $1/r$ of the time. What the conscience mechanism in SOMC does is to enforce this ratio of updates at all times. Let us change the Kohonen SOM algorithm as follows:

Let I_j be the indicator function which is 1 if centroid j is the closest to the input vector x, and 0 otherwise:

$$I_j = \begin{cases} 1 & \text{if } j = \min_k{}^{-1} [D(Y_k, x)] \\ 0 & \text{else} \end{cases}$$

In the case of a tie, $I_j = 1$ for the lower index j, and 0 for the higher.

Now, let p_j represent the fraction of time that processing element j wins the competition. One way to generate these numbers while suppressing random fluctuations in the input source is:

$$p_j(n+1) = p_j(n) + \mu[I_j - p_j(n)]$$

where $0 < \mu \ll 1$. The literature seems to indicate that values of μ around 0.0001 work well.

Now, let i be the winning processing element in terms of weight adjustment. A bias term b_k is introduced to modify the competition. Then,

$$i = \alpha(x) = \min_k{}^{-1} [D(Y_k, x) - b_k].$$

[37] It is shown in [47] that equidistortion regions are better than equiprobable regions. However, DeSienno defines his algorithm for equiprobable regions.

[38] Choosing other neighborhood functions can help mitigate this problem, but will never make it go away.

where,

$$b_k = \nu(1/N - p_k)$$

The constant ν represents the bias factor determines the distance a losing processing element can reach in order to enter the solution. By setting ν to zero, the algorithm becomes regular SOM learning. ν around 10 works for medium sized problems.

If the researcher wishes to maintain the projection of $f_\Phi(x)$ onto the 2D array of centroids, a neighborhood function may still be used in the weight update. The conscience mechanism has eliminated the need for it, however, if all that is required is the codebook of centroids itself, and not the spatial mapping.

8.4.4 LVQ: The Supervised Stochastic Algorithms

The following section describes the final class of vector quantizer learning algorithms, titled the "Learning Vector Quantization" (LVQ) algorithms. All of these algorithms update their centroids on a stochastic, sample by sample basis, and are supervised algorithms. The differences between the algorithms will be highlighted in the appropriate sections.

LVQ1

The first LVQ algorithm operates very similarly to the SOM algorithm, but with the exception that the desired response $\phi(x(n))$ of every input vector $x(n)$ is known during training. The "response" is chosen in the same way as in SOM:

$$i = \alpha(x) = \min_k {}^{-1} [D(Y_k, x)].$$

but, instead of simply moving that centroid toward x, we first check to see if $\phi(x(n))$ is identical to $\delta(i)$. If so, then centroid i is moved closer to x, else it is moved away from x:

$$\begin{aligned} Y_i(n+1) &= Y_i(n) + \xi(n)\,[x(n) - Y_i(n)] \\ &\quad \text{if } x \text{ and } Y_i \text{ are in the same class} \\ Y_i(n+1) &= Y_i(n) - \xi(n)\,[x(n) - Y_i(n)] \\ &\quad \text{if } x \text{ and } Y_i \text{ belong to different classes} \\ Y_k(n+1) &= Y_k(n) \text{ for } k \neq i \end{aligned}$$

As before, $0 < \xi(n) < 1$, and $\xi(n)$ may be constant or decrease monotonically with time. In the above basic LVQ1 it is recommended that ξ should initially be smaller than 0.1; linear decrease in time is favored.

OLVQ1

We now ask the question "Is there an optimal $\xi(n)$ to make the LVQ1 algorithm converge as quickly as possible?" In fact, if we assign different learning rates $\xi_k(n)$ to each centroid, there is. First, we re-express the weight update formula as:

$$Y_i(n+1) = [1 - s(n)\xi_i(n)]Y_i(n) + s(n)\xi_i(n)x(n) \quad (14)$$

$$Y_k(n+1) = Y_k(n) \text{ for } k \neq i \quad (15)$$

where $s(n)$ is the bipolar indicator function:

$$s(n) = \begin{cases} 1 & \text{if } \phi(x) = \delta(i) \\ -1 & \text{else} \end{cases} .$$

We see from (14) that $Y_i(n)$ is statistically independent of $x(n)$. It may also be obvious that the statistical accuracy of the learned codebook vector values is optimal if the effects of the corrections made at different times, when referring to the end of the learning period, are of equal weight. Notice that $Y_i(n+1)$ contains a "trace" from $x(n)$ through the last term in (14), and "traces" from the earlier $x(n'), n' = 1, 2, \ldots, n-1$ through $Y_i(n)$. The (absolute) magnitude of the last "trace" from $x(n)$ is scaled down by the factor $\xi_i(n)$, and, for instance, the "trace" from $x(n-1)$ is scaled down by $[1 - s(n)\xi_i(n)] \times \xi_i(n-1)$. Now we first stipulate that these two scalings must be identical:

$$\xi_i(n) = [1 - s(n)\xi_i(n)] \times \xi_i(n-1).$$

If this condition is then made to hold for all n, by induction it can be shown that the "traces" collected up to time n from all the earlier x will be scaled down by an equal amount at the end, and thus the "optimal" values of $\xi_i(n)$ are determined by the recursion

$$\xi_i(n+1) = \frac{\xi_i(n)}{1 + s(n)\xi_i(n)}.$$

In practice, this modification makes LVQ1 operate much faster. However, precaution must be taken (since $\xi_i(n)$ can rise as well as fall) that $\xi_i(n)$ never rise above the value 1, or the solution will become unstable. If we enforce that any ξ_i never rise above its initial value, we can select rather high initial values (say, 0.3) and have the algorithm work very rapidly.

OLVQ1 With a Conscience

While it is not reported anywhere in the literature (that this author is aware of), the similarity between the SOM algorithms and the LVQ1 algorithms lend weight to the idea of adding a conscience to LVQ1. Collecting the relevant formulae we get:

$$I_j = \begin{cases} 1 & \text{if } j = \min_k{}^{-1}[D(Y_k, x)] \\ 0 & \text{else} \end{cases}$$

$$
\begin{aligned}
p_j(n+1) &= p_j(n) + \mu[I_j - p_j(n)] \\
b_k &= \nu(1/N - p_k) \\
i &= \min_k{}^{-1}\,[D(Y_k, x) - b_k]. \\
s(n) &= \begin{cases} 1 & \text{if } \phi(x) = \delta(i) \\ -1 & \text{else} \end{cases} \\
\xi_i(n+1) &= \frac{\xi_i(n)}{1 + s(n)\xi_i(n)} \\
Y_i(n+1) &= [1 - s(n)\xi_i(n)]Y_i(n) + s(n)\xi_i(n)x(n)] \\
Y_k(n+1) &= Y_k(n) \text{ for } k \neq i
\end{aligned}
$$

This is not terribly burdensome in terms of computation, and may work quite well.

FSCL

One "conscience" method of supervised VQ learning which is reported in the literature is called "Frequency Sensitive Competitive Learning" or FSCL [2]. This algorithm is identical to LVQ1, except that the distortion criteria is modified as follows:

$$i = \alpha(x) = \min_k{}^{-1}\,[D(Y_k, x) \cdot f(k)].$$

where $f(k)$ is the cumulative number of times that centroid k has ever been updated. This distortion measure penalizes those centroids which have been updated most often. The rest of the algorithm is identical to that of LVQ1.

C&SL

The final distortion-based algorithm we will consider is one called "Competitive and Selective Learning". In [47], it is shown that the conscience methods based on finding equiprobable regions are sub-optimal in the sense of minimizing average distortion. It is shown that the optimal codebook will make the centroids have equi-partial-distortion. The C&SL algorithm performs this task.

The algorithm alternates between two phases. In the first phase, the centroids are trained using regular SOM learning, with no neighborhood function (for optimum distortion results, we must use unsupervised methods). This phase makes one pass through the training data, in random order, and updates the centroids as usual. After the training phase, the partial average distortions $D(k), k = 1 \ldots r$, of each centroid are computed. If the total average distortion has changed by less than ϵ, the algorithm terminates, else the "select" phase is run.

In the select phase, $s(m)$ centroids of the total r centroids are selected. Half of these centroids (rounded down) are those with highest partial distortion, and the other half

(rounded up) are those with lowest partial distortion. For these $s(m)$ selected centroids, we calculate:

$$g_j = \frac{\sqrt{D(j)}}{\sum_{j=1}^{s(m)} \sqrt{D(j)}}$$

Next, we compute:

$$u_j = \lfloor g_j s(m) \rfloor$$

where $\lfloor x \rfloor$ is the largest integer less than or equal to x. For the top $s(m) - \sum_{j=1}^{s(m)} u_j$ neurons with respect to the value $g_j s(m) - u_j$, add one to each u_j.

For every j, reproduce u_j neurons by adding random perturbation vectors to Y_j. We end up with $s(m)$ centroids as when we started, except that those centroids with the smallest partial distortion were replaced by near-copies of the centroids with highest partial distortion.

After the selection phase, the algorithm starts over at the training phase. Over time, a set of centroids with very similar average partial distortion is generated. At the end of the simulation, each centroid is classified according to the costs in Table 1.

The only unresolved issue is the determination of $s(m)$, the number of centroids selected at iteration number m of the algorithm. We have determined $s(m)$ to be initialized at $r/2$, and then decrease linearly with the iteration through the simulation.

Approximate Linear Decrease

Many of the VQ algorithms call for a linear decrease in the value of some parameter over the course of the simulation. Since the stopping criterion of the simulation is the difference in distortion between iterations, and not the iteration number, we do not know the number of iterations which will be performed, and thus we cannot directly calculate a linear decrease of some parameter. What we can do is to estimate the iteration that the simulation terminates on, and use this estimate to update the parameter.

The distortion of the codebook can be seen to decrease in a nearly exponential fashion versus the iteration of the VQ design algorithm. Therefore, we model this distortion as $A + B\exp[C \cdot m]$, where m is the iteration number, A is the final distortion of the simulation, $B - A$ is the excess distortion and C is the rate of decrease.

We compute:

$$\begin{aligned} Y &= (A + B\exp[C(m-1)]) - (A + B\exp[C(m)]) \\ &= B\exp[Cm]\exp[-C] - B\exp[CX] \\ &= B\left(\exp[-C] - 1\right)\exp[Cm] \\ Y' &= \ln Y \\ &= Cm + \ln\left(B(\exp[-C] - 1)\right) \end{aligned}$$

So, Y' is linear in m. We can use linear regression techniques to compute $\hat{Y}' =$

$a_1 m + a_0$. Then,

$$\begin{aligned} C &= a_1 \\ B &= \frac{\exp(a_0)}{\exp[-C] - 1} \end{aligned}$$

Our stopping criteria is:

$$\begin{aligned} Y &\leq \epsilon \\ B \exp[Cm] \left(\exp[-C] - 1\right) &\leq \epsilon \\ \exp[Cm] &\leq \frac{\epsilon}{B(\exp[-C] - 1)} \\ m &\geq \frac{1}{C} \ln \left[\frac{\epsilon}{B(\exp[-C] - 1)}\right] \end{aligned}$$

Using this estimate of the ending iteration, we can create a near-linear decrease in some parameter. After each iteration of training, the estimate of m is updated, and it becomes progressively better.

This method worked far better than other (parameterless & otherwise) methods of computing a decreasing function over time.

LVQ2.1

The LVQ2.1 algorithm examines a completely different approach to centroid generation from the other algorithms. The other algorithms presented place centroids to divide the input space into discrete, equiprobable regions. As such, they can be used to build up an estimate of the probability density functions and thus to distinguish the Bayesian decision boundary.

The LVQ2.1 and LVQ3 algorithm directly estimate the decision boundary with the centroids. In learning, two centroids are updated simultaneously. For each input $x(n)$, centroid Y_i and centroid Y_j may be updated. Centroid Y_i is the closest centroid to $x(n)$ which is of the same class as $x(n)$, and centroid Y_j is the closest centroid to $x(n)$ not of the same class.

$$\begin{aligned} i &= \min_k{}^{-1} \left[D(Y_k, x) | \delta(k) = \phi(x(n))\right] \\ j &= \min_k{}^{-1} \left[D(Y_k, x) | \delta(k) \neq \phi(x(n))\right] \end{aligned}$$

Furthermore, the update is only made if $x(n)$ falls within a small window surrounding the bisecting plane between centroid Y_i and Y_j. Let $d_i = D(Y_i, x(n))$ and $d_j = D(Y_j, x(n))$. Then, x(n) falls within the window of relative width 'w' if:

$$min\left(\frac{d_i}{d_j}, \frac{d_j}{d_i}\right) > \frac{1-w}{1+w}$$

widths of 0.2 to 0.3 work best.

The algorithm is:

$$\begin{aligned} Y_i(n+1) &= Y_i(n) + \xi(n)[x(n) - Y_i(n)] \\ Y_j(n+1) &= Y_j(n) - \xi(n)[x(n) - Y_j(n)] \end{aligned}$$

LVQ3

The LVQ2 algorithm was based on the idea of differentially shifting the decision borders towards the Bayes limits, while no attention was paid to what might happen to the location of the Y_k in the long run if this process were continued. Therefore it seems necessary to include corrections that ensure that the Y_k continue approximating the class distributions, at least roughly. Combining these ideas, we now obtain an improved algorithm that may be called LVQ3. If the two closest centroids to $x(n)$, Y_i and Y_j are of different classes, then:

$$\begin{aligned} Y_i(n+1) &= Y_i(n) + \xi(n)[x(n) - Y_i(n)] \\ Y_j(n+1) &= Y_j(n) - \xi(n)[x(n) - Y_j(n)] \end{aligned}$$

where $x(n)$ and Y_i belong to the same class, and $x(n)$ and Y_j belong to different classes, respectively; furthermore, $x(n)$ must fall within the window. If the two closest centroids are of the same class as $x(n)$ then,

$$Y_k(n+1) = Y_k(n) + \epsilon\xi(n)[x(n) - Y_k(n)]$$

for both $k \in \{i, j\}$. Values of ϵ between 0.1 and 0.5 seem to work, depending on the size of the window, being smaller for narrower windows. This algorithm seems to be self-stabilizing, i.e. the optimal placement of the Y_j does not change in continued learning.

DIFFERENCES BETWEEN THE BASIC LVQ1, LVQ2.1 and LVQ3.1

The three options for the LVQ-algorithms, namely, the LVQ1, the LVQ2.1 and the LVQ3, yield almost similar accuracies, although a different philosophy underlies each. The LVQ1 and the LVQ3 define a more robust process, whereby the codebook vectors assume stationary values even after extended learning periods. For the LVQ1 the learning rate can approximately be optimized for quick convergence with OLVQ1. In the LVQ2.1, the relative distances of codebook vectors from the class borders are optimized whereas there is no guarantee for the codebook vectors being placed optimally to describe the forms of the class borders. Therefore the LVQ2.1 should only be used in a differential fashion, using a small value of learning rate and a relatively low number of training steps.

8.4.5 Choosing Initial Codebooks

As has been mentioned earlier, codebook initialization is an important factor for locating a good set of centroids. Since all of the VQ design methods presented here are descent methods, the initial codebook will largely determine the final codebook. The stochastic methods have a chance of escaping some local minima since they can jump around more than the batch methods, but even these are affected by the initial choice.

It turns out that the methods used to initialize codebooks can be used to design codebooks themselves. Then, the VQ algorithms are used to optimize these codebooks. The following list presents a number of the algorithms in use today:

1. **Random Coding**: Centroids are assigned randomly according to the input distribution. The first r vectors drawn (from random, non-repeated positions) from the input distribution can be used (this is what has been used in our work). Since the mine lane data is correlated in space, the r vectors can not be drawn starting at the beginning, but must be drawn from random positions in the lane. Note that the centroids are first assigned randomly, but then are used deterministically as always.

2. **Lattice**: A regular lattice can be constructed which covers the input range evenly. Often, some sort of product code is used, where the cross product of k, where k is the dimension of the input space, scalar quantizers is used as the initial vector quantizer.

3. **Pruning**: With this method, we start with a Codebook which contains all of the training data, and "prune" out bad members until we have a good starting set. The pruning algorithm is:

```
begin {Pruning Algorithm}

        take first vector, add to codebook.
        repeat
                examine next vector.  If its distortion with respect to the
                    current codebook < ε, then discard it as a candidate.
                else add it to the codebook.
        until no more vectors, or codebook full
        if not enough centroids, decrease ε and repeat

end {Pruning Algorithm}
```

   This algorithm is similar to that used by ART. A good initial choice of ϵ is: $\epsilon = r^{-2/k}$ where r is the number of centroids desired, and k is the dimension of the input vectors.

4. **Pairwise Nearest Neighbor**: This method starts with a codebook containing all of the training vectors, and iteratively merges centroids until the desired number of centroids is reached. It works as follows:

begin {Pairwise NN Algorithm}

Set $\mathcal{C}$ = the training set.

Compute distortion between all pairs.

Merge the pair with lowest distortion, and replace with their centroid.

$$Y_{\text{new}} = \text{cent}(Y_{1,\text{old}}, Y_{2,\text{old}}).$$

repeat

For each pair of centroids, compute[a]:

$$\Delta_{i,j} = \sum_{l=1}^{L_i} D(x_i(l), \text{cent}(R_i)) + \sum_{l=1}^{L_j} D(x_j(l), \text{cent}(R_j)),$$

$$\Delta_{i,j}^{'} = \sum_{l=1}^{L_i} D(x_i(l), \text{cent}(R_i \bigcup R_j)) + \sum_{l=1}^{L_j} D(x_j(l), \text{cent}(R_j \bigcup R_j)) \geq \Delta_{i,j}.$$

Merge the pair of clusters R_i, R_j for which $\Delta_{i,j}^{'} - \Delta_{i,j}$ is smallest.

until the correct number of centroids is reached.

end {Pairwise NN Algorithm}

[a] The first is the contribution to the average distortion if their regions are not merged, and the second is the contribution if they are merged.

This is a "Greedy" algorithm: Each merge is optimal, but the entire procedure is not necessarily optimal.

5. **Splitting**: This procedure starts with a codebook which contains only the centroid of the entire training set. Then, iterations are performed to split the regions defined by each centroid until the desired number of centroids are attained.

begin {Splitting Algorithm}

Set $\mathcal{C} = \{\text{cent}(R)\}$.

repeat

For each centroid, split it as follows. First, compute the covariance matrix of $R(i)$. Compute the eigenvector v corresponding to the largest eigenvalue of this matrix. Then, set the two resulting centroids to:

$$Y_1 = Y_i + \epsilon v,$$

$$Y_2 = Y_i - \epsilon v,$$

Perform a Lloyd iteration on the region spanned by $R(i)$, to optimize the centroids Y_1 and Y_2.

until the correct number of centroids is reached.

end {Splitting Algorithm}

Note that only the *largest* eigenvalue needs to be computed. Rather than attempting to compute the entire set of eigenvalues and eigenvectors for a matrix which may be singular, the method called a "Power Method" may be used to do this:

```
begin {Power Method}
        Start with an estimate of v = v_0.
        repeat
                z_k = A v_{k-1}
                v_k = z_k / ||z_k||
                λ_k = v_k^H A v_k
        until |λ_k − λ_{k-1}| < ε.
end {Power Method}
```

where A is the covariance matrix, and λ is the largest eigenvalue. If v_0 has a component in the direction of v, then this method will converge on v. Also note that A is positive semi definite, and so all the eigenvalues will be real and non-negative, and the eigenvectors will also be real. Thus, the Hermitian transpose of v_k is simply a normal transpose.

6. **Simulated Annealing**: Another set of algorithms which can be used to find a globally optimal codebooks (or initial codebooks) are called "Simulated Annealing." The term 'Simulated Annealing' is descriptive since the procedure has a strong resemblance to the annealing of metals. In a molten metal, the atoms are in violent random motion. As with all physical systems, the atoms will naturally tend toward a minimum energy state (a solid metallic crystal), but at high temperatures the vigor of the atomic motions prevents this. As the metal is gradually cooled, lower and lower energy states are assumed until finally the lowest of all possible states, a global minimum is achieved.

   The simulated annealing algorithm closely emulates this physical procedure. The centroids are initially assigned randomly, and a high "temperature" is assumed. This allows large random centroid perturbation to occur at first, even if they make the distortion of the system increase. As the "temperature" is gradually lowered, the random centroid changes are allowed less often if they cause the distortion to increase, but are always allowed if the distortion decreases. When the temperature has reached zero, if the process has progressed slowly enough, then the final result will be the globally optimal solution. In practice, if the temperature decreases a little too rapidly, then a locally optimal solution is attained, which tends to be nearly as good as the globally optimal one.

   These statistical training methods are very robust and will always find a good solution. Their only drawback is computational speed; they may be very slow to converge. The Boltzmann learning method in particular is very slow; the Cauchy method is faster. Methods exist which use an "artificial specific heat" analogy to detect phase changes in the solution to help speed up the algorithm.

All of these algorithms can work in a supervised or unsupervised fashion. In the

unsupervised mode, the algorithm designs a codebook for all training vectors, and ignores the class of the vectors. In the supervised mode, the algorithm designs one set of codebooks for each input class. Only training vectors of class j are used to generate the codebook for class j. If we desire a final codebook of size r, then each of the smaller codebooks can be of size r/m if m is the number of classes (i.e. uniform size), or of $r \times p(j)$ if $p(j)$ is the *a priori* probability of class j (i.e. proportional size). There is reason to believe that the proportional method will work best, and this is what has been used in our work to date.

None of these methods generate codebooks which work when using any of the SOM algorithms. There are two basic methods which can be used:

1. **Random**: Initialize the centroids randomly.
2. **Lattice**: Initialize with a 2D lattice on the subspace spanned by the two principal eigenvectors of the correlation matrix of the training set.

The second method is probably the better method, although SOMC tends to make the centroids always converge on a good solution, even if the matrix is initialized with all zeros!

8.4.6 Optimizing the Functional Approximation vs. the Decision Boundary

Two different approaches were taken to the task of codebook generation. The majority of the methods tried to generate centroids which would accurately represent the pdf of the underlying data. The other methods, LVQ 2.1 and LVQ3, tried to represent the Bayesian decision boundaries instead. The question arises, "Which is best?"

There is no apparent answer to this question, but some observations will be made. First, for pattern recognition, all we care about are the decision boundaries. The distribution can be represented arbitrarily poorly; as long as the boundaries are accurate, we will perform Bayesian (optimal) classification.

However, we must consider that the VQ centroids are somehow used by the PNN or NN to perform pattern recognition. We know how a PNN works by its construction. It builds up an estimate of the posteriori probability density functions for each input class, from the input data, and performs classification based on which class has the highest posteriori probability. Therefore, the PNN requires an accurate mapping of the pdf and thus LVQ 2.1 and LVQ 3 probably will not give centroids which work well with a PNN.

The NN case is not obvious and would require experimentation. The NN is far more general than the PNN and is able to build up a highly nonlinear function of the input data. However, this function is based at least loosely on the distribution of input samples and probably will not work too well for LVQ 2.1 or LVQ 3 either.

LVQ 2.1 and LVQ 3 may work well as pattern classifiers if the NN is removed and the VQ performs the classification $\hat{\phi}(x)$. This may work well for the other methods too, but the NN is expected to perform better since it can compute highly non-linear functions, not just perform nearest neighbor classification. The NN's performance, $\hat{\phi}(\hat{x})$ may be enhanced,

however, by appending the VQ estimate $\hat{\phi}(x)$ to the input vector $\hat{x}$ to the NN when training, and to x when testing.

8.5 Deterministic Outlier Removal

When the data retained by the VQ outlier removal method was being computed, the location of those data points which were removed from the data set were plotted. An example of this is shown in Figure 58. The left side of the plot shows with "." the location of those vectors which were retained, and with "X" those vectors which were discarded. The right side of the plot shows the location of the mines, and their extended mine regions. We see from this figure, that many of the outlier points occur near the boundaries of the mines, where the accurate determination of the desired response is difficult. This observation supports the use of the deterministic outlier removal methods described in Section 6.

```
69: ........................... train ....=======--=.............
70: ........X.................. train ....=--111--=.............
71: ....XX....X................ train ....=-11111-=.............
72: .....X..................... train ....=-11111-=.............
73: .....X..................... train ....=-11111-=.............
74: ......X..X............XX... train ....=-11111-=......========
75: ......XXX..............X.. train ....=--------=......=------=
76: ......................XX.... train ....==============.=-2222-=
77: ..............XX.....X..... train ..........==-----=.=-2222-=
78: ..............X.....X...X. train ..........=--555-=.=-2222-=
79: ................X....X...X. train ..........=-5555-=.=--222-=
80: ................X.....XXX.X train .======...=-555--=.==------=
81: ............X..X........XX. train ==------==..=------==..=======
82: .....X........X............ train =--22--=..=======..........
83: ......X.................... train =-2222-=..................
84: .X......................... train =-2222-=..................
85: ........................... train =-2222-=......=========....
86: .....X...................... train =--22--=......=--------=....
87: ....................XXX..... train ==------==......=-44444-=....
88: ..............X..X..X..... train .======.......=-44444-=....
```

Figure 58: Sample VQ outlier location

9 PNN and VQ Results and Conclusions

The volume of results generated over the course of this contract is considerable, and it would not be worthwhile to duplicate them all here. Many of the early results were very poor; additionally, many of the results from previous reports were stated using different performance methodologies than is now used. Thus, only the most recent and most significant results are presented here.

The results are presented in order of complexity of the underlying classification device. Thus, the results from the nearest neighbor classifier are reported first, followed by the results from the PNN classifier. The results for the neural network classifier are presented last.

9.1 Nearest Neighbor Classification Results

A nearest neighbor classifier was built and runs were performed for benchmark cases, and using the centroids generated by the various VQ algorithms. As expected, the results are poorer than those found by a neural network classifier, but they are still useful in that they determine a lower bound for how well we would expect a classification system to work. The results are presented in Table 3, where the number to the left of each line indicates the number of VQ centroids used. The best results are extracted from this table and are presented in Table 2.

We see that the best VQ results are always for when the centroids themselves are used as a basis of comparison, and not for when the original data with the outliers removed is used. This result is in agreement with similar program runs for the PNN classification structure. As will be seen later, however, the NN gives better performance with the original data with VQ outliers removed. Thus, we conclude that the PNN classifier is inherently a nearest neighbor type of classifier (which is intuitively appealing given our knowledge of the PNN structure) and is fundamentally different in its operation from a NN classifier. While this would seem to cast doubt on the usefulness of a PNN classifier, it may in fact show that since the PNN and the NN operate in different ways, that even better results may be obtained by using them together. This is a topic requiring future research.

Flat Data w/	Training		Testing	
param = 1.0	% TP	% FP	% TP	% FP
VQ Algorithm				
LBG	100.0	0.050	85.4	3.296
OLVQ1	100.0	0.000	83.3	3.273
C&SL	100.0	0.297	81.2	3.132
FSCL	98.5	0.000	79.2	3.108
LVQ1	98.5	0.000	79.2	3.391
OLVQ1C	100.0	0.033	75.0	2.990
Benchmark	100.0	0.000	75.0	3.626

Table 2: Best Single Nearest Neighbor Classification Results

Flat Data w/ param = 1.0	Training		Testing	
	% TP	% FP	% TP	% FP
Benchmark Run				
Benchmark	100.0	0.000	75.0	3.626
LBG Runs (centroids)				
1000	100.0	0.066	75.0	3.014
1500	100.0	0.050	72.9	3.579
2000	100.0	0.033	75.0	2.990
2500	100.0	0.017	81.2	3.179
3000	**100.0**	**0.050**	**85.4**	**3.296**
3500	100.0	0.033	83.3	3.296
4000	100.0	0.033	83.3	3.202
LBG Runs (outliers removed)				
1000	100.0	0.000	75.0	3.720
1500	100.0	0.017	75.0	3.650
2000	100.0	0.017	75.0	3.650
2500	100.0	0.017	75.0	3.650
3000	**100.0**	**0.033**	**75.0**	**3.650**
3500	100.0	0.017	75.0	3.626
4000	100.0	0.017	75.0	3.626
LVQ1 Runs (centroids)				
1000	90.9	0.050	66.7	2.096
1500	97.0	0.050	75.0	3.108
2000	98.5	0.033	72.9	3.155
2500	**98.5**	**0.000**	**79.2**	**3.391**
3000	100.0	0.000	79.2	3.344
LVQ1 Runs (outliers removed)				
1000	89.4	0.033	64.6	2.590
1500	97.0	0.050	70.8	3.132
2000	97.0	0.033	70.8	3.085
2500	**98.5**	**0.000**	**75.0**	**3.485**
3000	98.5	0.000	75.0	3.508
OLVQ1 Runs (centroids)				
1000	92.4	0.050	66.7	2.708
1500	97.0	0.050	68.8	2.943
2000	98.5	0.033	75.0	2.990
2500	100.0	0.000	77.1	3.367
3000	**100.0**	**0.000**	**83.3**	**3.273**
OLVQ1 Runs (outliers removed)				
1000	90.9	0.033	68.8	2.637
1500	95.5	0.033	70.8	3.037
2000	97.0	0.033	70.8	3.155
2500	**98.5**	**0.000**	**75.0**	**3.532**
3000	98.5	0.000	75.0	3.532

Flat Data w/ param = 1.0	Training		Testing	
	% TP	% FP	% TP	% FP
OLVQ1C Runs (centroids)				
1000	98.5	0.000	75.0	3.532
1500	98.5	0.000	75.0	3.532
2000	98.5	0.000	75.0	3.532
2500	98.5	0.000	75.0	3.532
3000	**100.0**	**0.033**	**75.0**	**2.990**
OLVQ1C Runs (outliers removed)				
1000	**100.0**	**0.033**	**75.0**	**2.990**
1500	100.0	0.033	75.0	2.990
2000	100.0	0.033	75.0	2.990
2500	100.0	0.033	75.0	2.990
3000	100.0	0.033	75.0	2.990
FSCL Runs (centroids)				
1000	92.4	0.083	66.7	1.907
1500	98.5	0.000	68.8	2.637
2000	97.0	0.000	72.9	2.190
2500	100.0	0.017	77.1	3.202
3000	**98.5**	**0.000**	**79.2**	**3.108**
FSCL Runs (outliers removed)				
1000	92.4	0.050	70.8	2.684
1500	97.0	0.000	72.9	3.155
2000	97.0	0.000	70.8	2.990
2500	**100.0**	**0.017**	**75.0**	**3.532**
3000	98.5	0.000	75.0	3.603
C&SL Runs (centroids)				
1000	90.9	1.222	79.2	3.909
1500	100.0	0.776	79.2	4.756
2000	100.0	0.793	79.2	4.097
2500	**100.0**	**0.297**	**81.2**	**3.132**
3000	100.0	0.215	81.2	3.838
C&SL Runs (outliers removed)				
1000	90.9	0.215	66.7	2.849
1500	100.0	0.198	68.8	3.320
2000	**100.0**	**0.066**	**72.9**	**3.202**
2500	100.0	0.066	72.9	3.202
3000	100.0	0.017	68.8	3.461

Table 3: Single Nearest Neighbor Classification Results

9.2 VQ Sammon Maps

The ability to visualize the data is an important research tool. In our application, we have 88 dimensional vectors, and this is an impossibility. John W. Sammon Jr. [42] devised a non-linear projection from k dimensions to 2 dimensions, which retains the Euclidean distance between vectors. Patterns and clustering in the data will be evident in the mapping, and distances measured on the map between any two points corresponds to distances in k-space between those points. Regrettably, the algorithm requires storage of a $N \times N$ array, where N is the data set size. For our large data set, and available computing resources, this is infeasible.

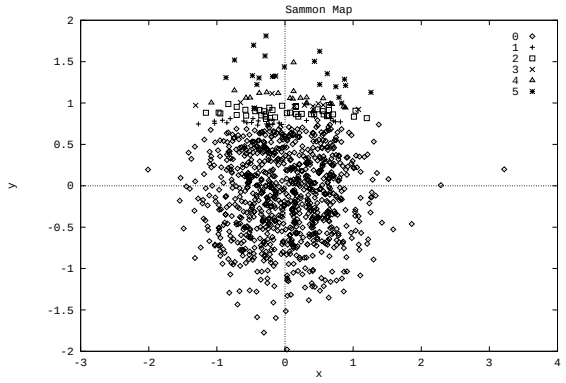
However, when the data set is reduced in size by the VQ centroid generating algorithms, we can use the Sammon mapping algorithm on the centroids. This was done, and Figure 59 shows the Sammon maps generated for the six VQ algorithms, where the maps were used for codebooks of size 1000 centroids. The legend at the top of each map explains what each point type is: background centroids are type 0, mine 1 centroids are type 1, and so on.

We can see from the shape of the maps that some VQ centroid generation algorithms work better than others. For all the algorithms except for the C&SM algorithm, the mine data points are on top of the figure, and the background data points are below them. We can see here that the mine data is quite easily separated from the background data. The points produced by the LBG algorithm are separated the best. Those produced by the C&SM algorithm, (which is the only algorithm trained to minimize distortion in an unsupervised manner) are the worst separated as it returns some mine centroids in the middle of the cloud of background centroids. These two facts combined indicate that there are mine patterns in the region occupied mostly by background patterns, but that they are infrequent. Most of the mine patterns are distinct from the background patterns. This observation encourages us that a solution to the mine lane classification problem with low error rate exists.

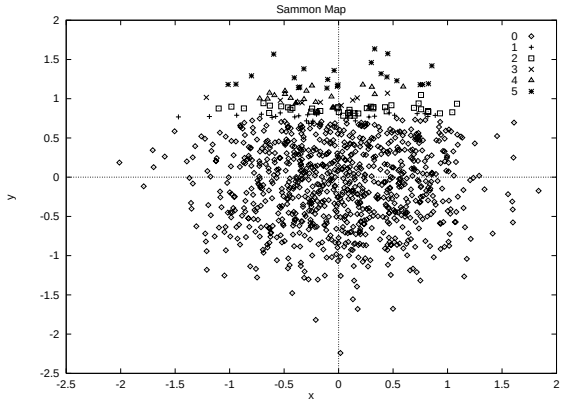
9.3 PNN & Transform Results

The probabilistic neural network has been used for some time as an alternate solution to neural network for the mine detection problem. Many sets of runs have been performed with the PNN architecture, including VQ runs [19]. However, the VQ runs were performed using a different method of reporting results than is currently used. Thus, only qualitative remarks can be made. It was noted that centroids generated by supervised VQ algorithms worked much better than centroids generated by unsupervised algorithms. This is in agreement with the NN results. It was also noted that results of runs which used the centroids themselves as the PNN weights were much better than the results of runs which used the mine lane data with the outliers removed as PNN weights. This is in agreement with the nearest neighbor classifier, but is different from the NN classifier.

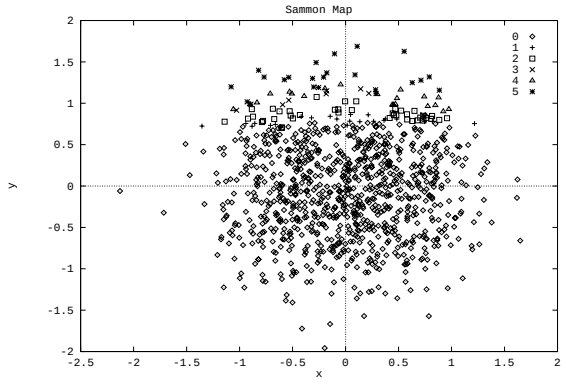
The following sections describe the results from using the PNN in conjunction with



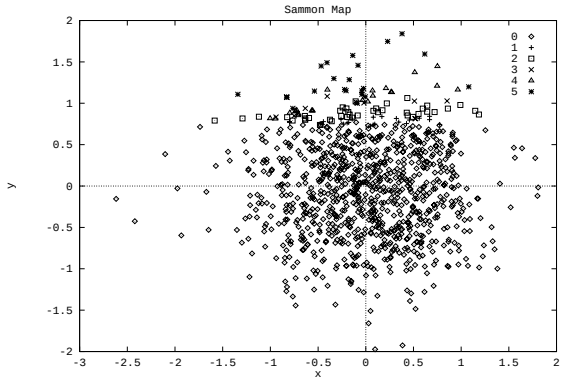
LBG Algorithm



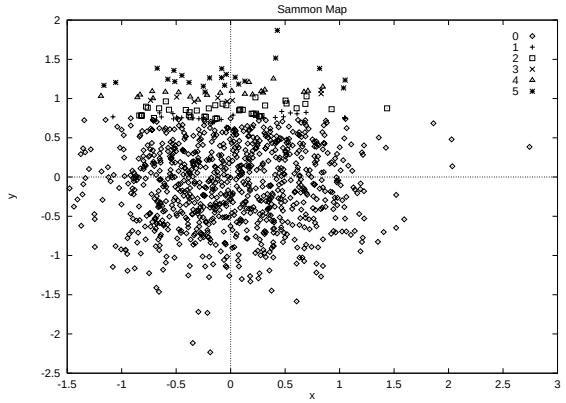
LVQ1 Algorithm



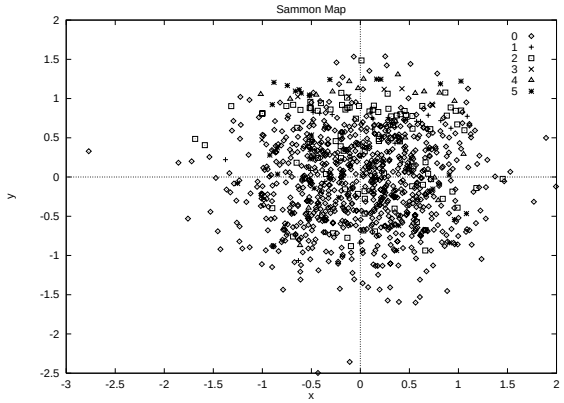
OLVQ1 Algorithm



OLVQ1C Algorithm



FSCL Algorithm



C&SL Algorithm

Figure 59: Sammon Maps for 1000 Centroids & 6 VQ Algorithms

the various transform methods. First, some results pertaining to the ESM algorithm are presented, followed by results from PNN simulations.

All simulations were performed with the PNN network, and used an extended mine region of 1. For the KL based methods, several different energy levels were simulated for each scenario. We note that there is no direct correspondence between MSE in the reconstruction and the probability of classification error. Thus, there is no real trend in which energy level is optimal for any of the transforms. In most cases, any level tested worked approximately equally well. In all of the tables, the best result is highlighted in bold face font. The best results for each transform paradigm are also summarized in Section 9.4.

9.3.1 ESM Results

The eigenspace separation algorithm is designed to create a distinction in the norms of the data vectors depending on their class. A plain orthonormal linear transformation preserves vector lengths, and is only able to rotate points in the input space. The ESM algorithm performs an orthonormal linear transformation and then discards coefficients from the transformed vector in such a way that those coefficients thrown away will cause maximal differentiation in the norms of the two classes. Conceivably, the best linear transformation is still unable to allow good separation. To test the ESM algorithm's separating ability, some tests were run on the mine lane data.

For data in the training section[39]:

Without ESM: The average mine square norm is 98.2707
The average background square norm is 41.4149
The difference is 56.8558

With ESM: The average mine square norm is 88.2821
The average background square norm is 27.5359
The difference is 60.7462 = E_p

For data in the entire mine lane (training and testing section):

Without ESM: The average mine square norm is 83.8602
The average background square norm is 38.6237
The difference is 45.2365

With ESM: The average mine square norm is 73.2388
The average background square norm is 25.3297
The difference is 47.9091

[39] We see that the separation in the norms is equal to E_p as predicted by theory. This was one of many tests performed to test the operational correctness of the simulator.

We see an improvement in the separation of square norms for both the training section and the entire mine lane, but the improvement is fairly small. Since, by Bayes' rule, we know that the best classification we can do depends on the probability density functions of the data and not necessarily on the separation of the means, we plot the histograms of the vector square norms of the data in Figure 60. In the histograms we see that the one-dimensional *pdf*s based on the square norms do not change much (the top four histograms are for the training region, and the bottom four histograms are for the entire mine lane. Note that the histograms are in terms of vector norms and not in terms of the square of vector norms.) Perhaps a nonlinear separation algorithm could separate the means and even the density functions better than the ESM algorithm which is constrained to be linear.

9.3.2 Benchmark Runs

To provide a basis for comparison, benchmark runs were performed with no transform methods applied, with each of the various preprocessing methods. Runs were performed both with and without the (single) 4-NN outlier removal method, and the results are tabulated in Tables 4 and 5.

In both scenarios, the simulations with Unit Norm preprocessing provided better results than without the Unit Norm preprocessing, although with a penalty in the false positive rate. Outlier removal caused an additional increase in the false positive rate (which is not surprising), and no net increase in the true positive rate. It was a general trend in the results of the simulations that 4-NN outlier removal added nothing to the results, except if Zero Mean preprocessing was performed as well. Simulations with 4-NN but without Zero Mean tended to be worse than if 4-NN was not used at all. It is possible that 4-NN outlier removal is removing some significant points from the training data.

9.3.3 The Eigenspace Separation Method

Tables 6 to 9 present the results for the Eigenspace Separation Method runs. The number of components retained in each case is reported as [#/44], where # is positive if the first # components from the matrix were used for the transform, and negative if the last |#| components were used. In all cases, Unit Norm preprocessing produced the best results. For the regular runs, the UNKL transform helped in some cases, and hindered in others, and overall seems to add nothing. For the 4-NN outlier removed cases, the UNKL transform provided a uniform improvement in the number of true positives, with a corresponding increase in the number of false positives.

9.3.4 The KL Transform Method

Tables 10 to 13 present the results for the KL transform runs. Runs were performed retaining 80%-95% of the energy of the original *pdf*, and the corresponding number of components

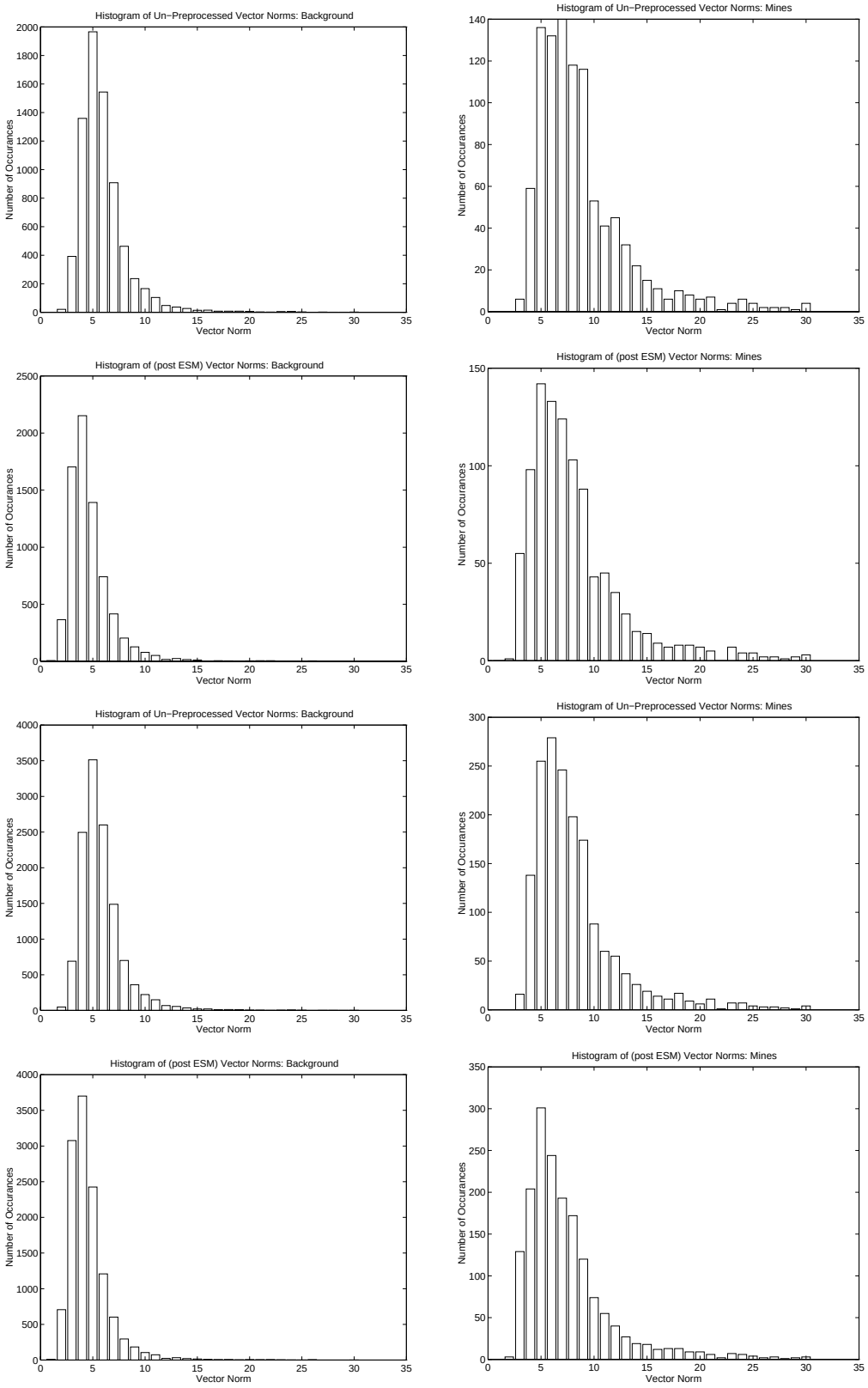


Figure 60: Histograms of Vector Norms: Before and after ESM processing; Background and Mine Patterns. The top four histograms are for the test region of the data, and the bottom four histograms are for the training region of the data.

retained from the KL transform (out of a total of 44) is reported next to the simulation name as [#/44].

The best overall results came from the whitened KL runs. This is predicted by theory, since the PNN basis functions are radially symmetric, and the whitened covariance matrix will also be approximately radially symmetric. Thus, the basis functions approximate the underlying *pdf* well. This result may not carry over to the NN, which is capable of performing more general decision boundaries with greater ease.

Again, all results with Unit Norm preprocessing performed better than comparative runs without it. We notice that the UNKL transform produces results that are almost identical (in some cases exactly identical) to the runs without it. We conclude that it helps very little, if at all, while adding complexity to the system. Thus, it is probably undesirable to pair the KL/UNKL transforms. As before, outlier removal added to the performance only for the Zero Mean preprocessing simulations.

9.3.5 The KL+ Transform Method

Tables 14 to 17 present the results for the KL+ transform runs. Again, very good results came from the whitened KL+ runs. They were bettered by a regular KL+ run with outlier removal, but only in the false positive rate. So, we conclude that whitening the KL type transforms is probably a good idea (at least when using PNN).

Results with Unit Norm preprocessing generally (but not uniformly) performed better than comparative runs without it.

Remarkably, we notice that the UNKL transform produces results that are exactly identical to the runs without it (these results were double checked, and are accurate). Thus, UNKL adds nothing in this case to the mine detection capability of the system, while adding significant complexity to the system.

As before, outlier removal added most to the performance for the Zero Mean preprocessing simulations.

9.3.6 The SDM Transform Method

Tables 18 to 21 present the results for the Simultaneous Diagonalization Method Method runs. Runs were performed retaining, as a first order approximation, 80%-95% of the energy of the original *pdf*, and the corresponding number of components retained from the KL transform (out of a total of 44) is reported next to the simulation name as [#/44].

The best overall results came from the zero mean preprocessing results. In fact, the zero mean runs had the highest TP rate with a very low FP rate. The unit norm results are good, but are not the best results, as are the unit norm results for the other transform methods.

9.3.7 The Composite Transform Method

Table 22 presents the results for the composite transform runs. The number of components retained from the KL_+, KL_- transform matrices is reported as [#KL_+,#KL_-]. The only cases simulated considered no Zero Mean preprocessing. For this transformation method, outlier removal provided the best results for both the unpreprocessed and Unit Norm preprocessed runs. For the unit norm runs, UNKL also improved performance. We note that the Unit Norm runs were always at least as good as the unpreprocessed runs (although with higher false positives).

9.4 Summary PNN Results

9.4.1 No Preprocessing

Table 23 summarizes the best[40] (un-preprocessed) results from all of the simulation categories. We see, that for no preprocessing, all of the transform methods performed better than the benchmark runs. The whitened KL runs performed best of all, and had a good FP level too.

9.4.2 Lossy Unit Norm Preprocessing

Table 24 summarizes the best (unit norm preprocessed) results from all of the simulation categories. Again, all of the transform methods performed better than the benchmark runs. Again, the whitened KL runs performed best of all, and had a good FP level too. The results here are better than the results for no preprocessing as well. The SDM run has an equivalent TP rate as whitened KL, but a high FP rate.

9.4.3 Lossy Zero-Mean Preprocessing

Table 25 summarizes the best (zero mean preprocessed) results from all of the simulation categories. Again, all of the transform methods performed better than the benchmark runs. We also notice that the FP rate is, on average, lowest for this type of preprocessing. Here, the SDM runs perform best, and the best results overall are the SDM runs here.

9.4.4 Lossy Zero-Mean and Unit Norm Preprocessing

Table 26 summarizes the best (zero mean and unit norm preprocessed) results from all of the simulation categories. Here, the whitened KL+ transform provided the best results, with 4-NN KL approximately the same. The ESM and SDM runs are also approximately the same, with the SDM algorithm having a slightly higher FP rate.

[40] As always, the "best" results are those with the highest rate of true positives with the lowest false positive rate in the testing section. In the case of a tie, the result with the highest rate of true positives with the lowest rate of false positives in the training section is chosen.

9.4.5 General Conclusions

1. Two of the four best results came from Table 13. This lends credence to the idea that whitening the KL transform by dividing by the square root of the eigenvalues is a profitable exercise. Perhaps this is only true for the PNN which estimates the *pdf* using radial functions; a whitened covariance matrix is easier to estimate with radial functions than an elliptical covariance matrix.

2. For most of the transforms, unit norm preprocessing provided the best results. However, if 4-NN outlier removal is used, zero mean preprocessing gives the best results. Also, for the SDM transformation, zero mean preprocessing gives the best results.

3. KL+ offers no real benefit, and should be discarded.

4. By noticing the number of tables referenced per summary result, and the number of simulations run (varying the number of transformed coefficients retained) per table we can see that each summary result for the KL case is the best of 8 runs, each result for KL+ is the best of 12 runs, each ESM result is the best of 2 runs, each composite run is the best of 1 run and each SDM result is the best of 12 runs. Therefore, we might conclude that some of the transforms were given "more chances" than the others to yield a good result. What is needed is a standard method of computing the number of coefficients retained by the KL related transform methods in order to perform fewer simulations and give an even playing field. An observation made during the simulation runs was that the optimum number of coefficients retained for each of the transforms was roughly two times the number of eigenvalues that were greater than the average eigenvalue. i.e. $m = 2||\{\lambda_i : \lambda_i \geq \overline{\lambda}\}||$. This hypothesis was not tested, and is a topic requiring future research.

5. ESM almost always had the lowest FP rate, but also always had the worst TP rate of the transform methods.

6. All of the transform methods provided better results than the benchmark runs.

Flat Data w/ param = 1.0	PNN σ ×1000	Training		Testing	
		% TP	% FP	% TP	% FP
No Preprocessing	400	100.0	0.0	75.0	3.579
Unit Norm Preprocessing	90	**100.0**	**0.0**	**85.4**	**5.604**
Zero Mean Preprocessing	500	100.0	0.0	77.1	3.508
Zero Mean and Unit Norm Preprocessing	80	100.0	0.0	83.3	5.580

Table 4: Benchmark PNN Runs

Flat Data w/ param = 1.0	PNN σ ×1000	Training		Testing	
		% TP	% FP	% TP	% FP
No Preprocessing	600	100.0	0.0	77.1	4.850
Unit Norm Preprocessing	60	100.0	0.0	81.2	7.558
Zero Mean Preprocessing	350	100.0	0.0	77.1	4.332
Zero Mean and Unit Norm Preprocessing	100	**100.0**	**0.0**	**83.3**	**7.864**

Table 5: Benchmark, 4-NN PNN Runs

Flat Data w/ param = 1.0	PNN σ ×1000	Training		Testing	
		% TP	% FP	% TP	% FP
No Preprocessing, ESM: [24/44]	600	100.0	0.0	79.2	2.543
Unit Norm Preprocessing, ESM: [-33/44]	30	**100.0**	**0.0**	**85.4**	**3.132**
Zero Mean Preprocessing, ESM: [25/44]	700	100.0	0.0	77.1	1.672
Zero Mean and Unit Norm Preprocessing, ESM: [-32/44]	30	100.0	0.0	83.3	3.555

Table 6: ESM Transformation Runs

Flat Data w/ param = 1.0	PNN σ ×1000	Training		Testing	
		% TP	% FP	% TP	% FP
No Preprocessing, ESM: [24/44]	600	100.0	0.0	79.2	2.355
Unit Norm Preprocessing, ESM: [-33/44]	80	**100.0**	**0.0**	**85.4**	**3.650**
Zero Mean Preprocessing, ESM: [25/44]	700	100.0	0.0	79.2	1.648
Zero Mean and Unit Norm Preprocessing, ESM: [-32/44]	60	100.0	0.0	81.2	3.391

Table 7: ESM Transformation / UNKL Runs

Flat Data w/ param = 1.0	PNN σ ×1000	Training		Testing	
		% TP	% FP	% TP	% FP
No Preprocessing					
No Preprocessing, ESM: [24/44]	600	100.0	0.0	79.2	3.296
Unit Norm Preprocessing, ESM: [-33/44]	70	**100.0**	**0.0**	**81.2**	**4.191**
Zero Mean Preprocessing, ESM: [25/44]	300	100.0	0.0	79.2	3.862
Zero Mean and Unit Norm Preprocessing, ESM: [-32/44]	70	100.0	0.0	79.2	4.756

Table 8: ESM Transformation, 4-NN Runs

Flat Data w/ param = 1.0	PNN σ ×1000	Training		Testing	
		% TP	% FP	% TP	% FP
No Preprocessing, ESM: [24/44]	600	100.0	0.0	79.2	3.202
Unit Norm Preprocessing, ESM: [-33/44]	50	**100.0**	**0.0**	**87.5**	**5.086**
Zero Mean Preprocessing, ESM: [25/44]	300	100.0	0.0	79.2	3.603
Zero Mean and Unit Norm Preprocessing, ESM: [-32/44]	50	100.0	0.0	87.5	5.416

Table 9: ESM Transformation, 4-NN / UNKL Runs

Flat Data w/ param = 1.0	PNN σ ×1000	Training % TP	Training % FP	Testing % TP	Testing % FP
No Preprocessing					
KL: 80% [10/44]	500	100.0	0.0	79.2	3.956
KL: 85% [12/44]	300	100.0	0.0	72.9	4.144
KL: 90% [15/44]	300	**100.0**	**0.0**	**83.3**	**3.485**
KL: 95% [21/44]	400	100.0	0.0	79.2	3.862
Unit Norm Preprocessing					
KL: 80% [12/44]	80	100.0	0.0	81.2	5.298
KL: 85% [14/44]	50	100.0	0.0	85.4	4.803
KL: 90% [17/44]	60	100.0	0.0	85.4	4.756
KL: 95% [22/44]	120	**100.0**	**0.0**	**87.5**	**5.062**
Zero Mean Preprocessing					
KL: 80% [11/44]	300	100.0	0.0	75.0	4.285
KL: 85% [13/44]	600	**100.0**	**0.0**	**79.2**	**2.896**
KL: 90% [16/44]	300	100.0	0.0	79.2	3.603
KL: 95% [21/44]	500	100.0	0.0	79.2	3.485
Zero Mean and Unit Norm Preprocessing					
KL: 80% [12/44]	40	100.0	0.0	85.4	5.416
KL: 85% [14/44]	60	100.0	0.0	85.4	5.510
KL: 90% [17/44]	60	100.0	0.0	83.3	5.439
KL: 95% [22/44]	80	**100.0**	**0.0**	**87.5**	**5.557**

Table 10: KL Transform Runs

Flat Data w/ param = 1.0	PNN σ ×1000	Training % TP	Training % FP	Testing % TP	Testing % FP
No Preprocessing					
KL: 80% [10/44]	350	100.0	0.0	75.0	6.546
KL: 85% [12/44]	300	100.0	0.0	72.9	5.934
KL: 90% [15/44]	450	100.0	0.0	79.2	5.910
KL: 95% [21/44]	350	**100.0**	**0.0**	**79.2**	**4.709**
Unit Norm Preprocessing					
KL: 80% [12/44]	140	100.0	0.2	81.2	6.381
KL: 85% [14/44]	50	100.0	0.0	85.4	7.558
KL: 90% [17/44]	120	100.0	0.0	83.3	7.158
KL: 95% [22/44]	120	**100.0**	**0.0**	**85.4**	**7.370**
Zero Mean Preprocessing					
KL: 80% [11/44]	300	**100.0**	**0.0**	**87.5**	**5.910**
KL: 85% [13/44]	300	100.0	0.0	85.4	5.204
KL: 90% [16/44]	350	100.0	0.0	83.3	5.204
KL: 95% [21/44]	350	100.0	0.0	79.2	4.733
Zero Mean and Unit Norm Preprocessing					
KL: 80% [12/44]	40	100.0	0.0	87.5	7.535
KL: 85% [14/44]	60	100.0	0.0	89.6	7.747
KL: 90% [17/44]	60	**100.0**	**0.0**	**89.6**	**6.899**
KL: 95% [22/44]	60	100.0	0.0	87.5	7.346

Table 11: KL Transform, 4-NN Runs

Flat Data w/ param = 1.0	PNN σ ×1000	Training		Testing	
		% TP	% FP	% TP	% FP
No Preprocessing					
KL: 80% [10/44]	250	100.0	0.0	79.2	4.168
KL: 85% [12/44]	300	100.0	0.0	72.9	4.144
KL: 90% [15/44]	300	**100.0**	**0.0**	**83.3**	**3.485**
KL: 95% [21/44]	400	100.0	0.0	79.2	3.862
Unit Norm Preprocessing					
KL: 80% [12/44]	100	100.0	0.0	81.2	5.109
KL: 85% [14/44]	50	100.0	0.0	85.4	4.803
KL: 90% [17/44]	50	100.0	0.0	85.4	4.591
KL: 95% [22/44]	120	**100.0**	**0.0**	**87.5**	**5.062**
Zero Mean Preprocessing					
KL: 80% [11/44]	300	100.0	0.0	75.0	4.285
KL: 85% [13/44]	600	**100.0**	**0.0**	**79.2**	**2.896**
KL: 90% [16/44]	300	100.0	0.0	79.2	3.603
KL: 95% [21/44]	500	100.0	0.0	79.2	3.485
Zero Mean and Unit Norm Preprocessing					
KL: 80% [12/44]	50	100.0	0.0	85.4	5.557
KL: 85% [14/44]	50	100.0	0.0	85.4	5.392
KL: 90% [17/44]	50	100.0	0.0	83.3	5.345
KL: 95% [22/44]	80	**100.0**	**0.0**	**87.5**	**5.557**

Table 12: KL Transform / UNKL Runs

Flat Data w/ param = 1.0	PNN σ ×1000	Training		Testing	
		% TP	% FP	% TP	% FP
No Preprocessing					
KL: 80% [10/44]	150	100.0	0.0	77.1	3.508
KL: 85% [12/44]	300	100.0	0.0	77.1	4.332
KL: 90% [15/44]	400	**100.0**	**0.0**	**87.5**	**3.414**
KL: 95% [21/44]	500	100.0	0.0	79.2	3.273
Unit Norm Preprocessing					
KL: 80% [12/44]	530	100.0	0.0	81.2	4.544
KL: 85% [14/44]	200	**100.0**	**0.0**	**91.7**	**5.062**
KL: 90% [17/44]	230	100.0	0.0	83.3	5.180
KL: 95% [22/44]	330	100.0	0.0	89.6	7.511
Zero Mean Preprocessing					
KL: 80% [11/44]	300	100.0	0.0	72.9	4.121
KL: 85% [13/44]	300	**100.0**	**0.0**	**83.3**	**3.414**
KL: 90% [16/44]	400	100.0	0.0	75.0	3.202
KL: 95% [21/44]	450	100.0	0.0	77.1	3.650
Zero Mean and Unit Norm Preprocessing					
KL: 80% [12/44]	160	100.0	0.0	81.2	4.591
KL: 85% [14/44]	200	**100.0**	**0.0**	**87.5**	**5.157**
KL: 90% [17/44]	240	100.0	0.0	87.5	5.274
KL: 95% [22/44]	310	100.0	0.0	85.4	6.428

Table 13: Whitened KL Transform Runs

Flat Data w/ param = 1.0	PNN σ ×1000	Training		Testing	
		% TP	% FP	% TP	% FP
No Preprocessing					
KL+: 80% [8/44]	300	**100.0**	**0.0**	**81.2**	**4.733**
KL+: 85% [10/44]	500	100.0	0.0	75.0	4.427
KL+: 90% [12/44]	500	100.0	0.0	75.0	3.956
KL+: 95% [16/44]	300	100.0	0.0	72.9	3.367
Unit Norm Preprocessing					
KL+: 80% [9/44]	80	**100.0**	**0.0**	**83.3**	**4.003**
KL+: 85% [11/44]	80	100.0	0.0	75.0	4.380
KL+: 90% [14/44]	40	100.0	0.0	81.2	3.767
KL+: 95% [19/44]	100	100.0	0.0	81.2	4.121
Zero Mean Preprocessing					
KL+: 80% [8/44]	350	**100.0**	**0.0**	**83.3**	**4.285**
KL+: 85% [10/44]	450	100.0	0.0	70.8	3.603
KL+: 90% [13/44]	400	100.0	0.0	77.1	3.697
KL+: 95% [16/44]	500	100.0	0.0	72.9	2.967
Zero Mean and Unit Norm Preprocessing					
KL+: 80% [10/44]	100	100.0	0.0	77.1	5.486
KL+: 85% [12/44]	60	100.0	0.0	79.2	5.204
KL+: 90% [15/44]	40	100.0	0.0	79.2	3.650
KL+: 95% [19/44]	140	**100.0**	**0.0**	**85.4**	**4.450**

Table 14: KL+ Transform Runs

Flat Data w/ param = 1.0	PNN σ ×1000	Training		Testing	
		% TP	% FP	% TP	% FP
No Preprocessing					
KL+: 80% [8/44]	350	100.0	0.0	81.2	6.758
KL+: 85% [10/44]	650	100.0	0.1	77.1	5.769
KL+: 90% [12/44]	300	**100.0**	**0.0**	**81.2**	**5.675**
KL+: 95% [16/44]	350	100.0	0.0	75.0	5.580
Unit Norm Preprocessing					
KL+: 80% [9/44]	100	100.0	0.1	79.2	4.874
KL+: 85% [11/44]	40	100.0	0.0	79.2	5.463
KL+: 90% [14/44]	60	100.0	0.0	77.1	6.357
KL+: 95% [19/44]	60	**100.0**	**0.0**	**81.2**	**6.051**
Zero Mean Preprocessing					
KL+: 80% [8/44]	250	**100.0**	**0.0**	**89.6**	**5.910**
KL+: 85% [10/44]	500	100.0	0.0	83.3	4.474
KL+: 90% [13/44]	300	100.0	0.0	85.4	5.628
KL+: 95% [16/44]	300	100.0	0.0	75.0	5.321
Zero Mean and Unit Norm Preprocessing					
KL+: 80% [10/44]	60	100.0	0.0	85.4	6.334
KL+: 85% [12/44]	40	100.0	0.0	81.2	6.193
KL+: 90% [15/44]	60	100.0	0.0	81.2	5.651
KL+: 95% [19/44]	60	**100.0**	**0.0**	**85.4**	**6.004**

Table 15: KL+ Transformation, 4-NN Runs

Flat Data w/ param = 1.0	PNN σ ×1000	Training		Testing	
		% TP	% FP	% TP	% FP
No Preprocessing					
KL+: 80% [8/44]	300	**100.0**	**0.0**	**81.2**	**4.733**
KL+: 85% [10/44]	500	100.0	0.0	75.0	4.427
KL+: 90% [12/44]	500	100.0	0.0	75.0	3.956
KL+: 95% [16/44]	300	100.0	0.0	72.9	3.367
Unit Norm Preprocessing					
KL+: 80% [9/44]	80	**100.0**	**0.0**	**83.3**	**4.003**
KL+: 85% [11/44]	80	100.0	0.0	75.0	4.380
KL+: 90% [14/44]	40	100.0	0.0	81.2	3.767
KL+: 95% [19/44]	100	100.0	0.0	81.2	4.121
Zero Mean Preprocessing					
KL+: 80% [8/44]	350	**100.0**	**0.0**	**83.3**	**4.285**
KL+: 85% [10/44]	450	100.0	0.0	70.8	3.603
KL+: 90% [13/44]	400	100.0	0.0	77.1	3.697
KL+: 95% [16/44]	500	100.0	0.0	72.9	2.967
Zero Mean and Unit Norm Preprocessing					
KL+: 80% [10/44]	100	100.0	0.0	77.1	5.486
KL+: 85% [12/44]	60	100.0	0.0	79.2	5.204
KL+: 90% [15/44]	40	100.0	0.0	79.2	3.650
KL+: 95% [19/44]	140	**100.0**	**0.0**	**85.4**	**4.450**

Table 16: KL+ Transformation / UNKL Runs

Flat Data w/ param = 1.0	PNN σ ×1000	Training		Testing	
		% TP	% FP	% TP	% FP
No Preprocessing					
KL+: 80% [8/44]	200	100.0	0.0	68.8	3.249
KL+: 85% [10/44]	200	100.0	0.0	72.9	3.744
KL+: 90% [12/44]	250	100.0	0.0	77.1	3.885
KL+: 95% [16/44]	250	**100.0**	**0.0**	**77.1**	**3.814**
Unit Norm Preprocessing					
KL+: 80% [9/44]	160	**100.0**	**0.0**	**87.5**	**4.521**
KL+: 85% [11/44]	220	100.0	0.0	83.3	5.628
KL+: 90% [14/44]	590	100.0	0.0	83.3	4.803
KL+: 95% [19/44]	310	100.0	0.0	87.5	7.229
Zero Mean Preprocessing					
KL+: 80% [8/44]	100	100.0	0.0	72.9	3.838
KL+: 85% [10/44]	150	100.0	0.0	70.8	3.697
KL+: 90% [13/44]	300	100.0	0.0	75.0	2.873
KL+: 95% [16/44]	250	**100.0**	**0.0**	**85.4**	**3.720**
Zero Mean and Unit Norm Preprocessing					
KL+: 80% [10/44]	140	100.0	0.0	83.3	6.051
KL+: 85% [12/44]	540	100.0	0.0	85.4	4.332
KL+: 90% [15/44]	340	100.0	0.0	87.5	5.227
KL+: 95% [19/44]	310	**100.0**	**0.0**	**89.6**	**6.664**

Table 17: Whitened KL+ Transform Runs

Flat Data w/ param = 1.0	PNN σ ×1000	Training % TP	Training % FP	Testing % TP	Testing % FP
No Preprocessing					
SDM: 80% [29/44]	600	100.0	0.0	75.0	2.519
SDM: 85% [32/44]	600	100.0	0.0	68.8	3.132
SDM: 90% [36/44]	650	100.0	0.0	75.0	2.519
SDM: 95% [39/44]	600	**100.0**	*0.0*	**77.1**	**3.202**
Unit Norm Preprocessing					
SDM: 80% [32/44]	400	100.0	0.0	83.3	6.852
SDM: 85% [35/44]	450	**100.0**	**0.0**	**89.6**	**6.758**
SDM: 90% [38/44]	400	100.0	0.0	87.5	6.805
SDM: 95% [41/44]	550	100.0	0.0	89.6	8.241
Zero Mean Preprocessing					
SDM: 80% [30/44]	550	100.0	0.0	89.6	3.767
SDM: 85% [33/44]	600	100.0	0.0	87.5	3.579
SDM: 90% [36/44]	600	**100.0**	**0.0**	**91.7**	**3.249**
SDM: 95% [40/44]	750	100.0	0.0	83.3	3.085
Zero Mean and Unit Norm Preprocessing					
SDM: 80% [31/44]	500	**100.0**	**0.0**	**85.4**	**4.780**
SDM: 85% [34/44]	450	100.0	0.0	83.3	5.251
SDM: 90% [37/44]	450	100.0	0.0	83.3	5.792
SDM: 95% [40/44]	850	100.0	0.0	83.3	6.499

Table 18: SDM Transformation Runs

Flat Data w/ param = 1.0	PNN σ ×1000	Training % TP	Training % FP	Testing % TP	Testing % FP
No Preprocessing					
SDM: 80% [28/44]	500	**100.0**	**0.0**	**83.3**	**3.862**
SDM: 85% [31/44]	650	100.0	0.0	81.2	2.543
SDM: 90% [35/44]	750	100.0	0.0	79.2	2.567
SDM: 95% [39/44]	550	100.0	0.0	81.2	3.579
Unit Norm Preprocessing					
SDM: 80% [32/44]	400	100.0	0.0	89.6	9.395
SDM: 85% [35/44]	450	**100.0**	**0.0**	**91.7**	**10.313**
SDM: 90% [38/44]	450	100.0	0.0	87.5	8.382
SDM: 95% [41/44]	700	100.0	0.0	89.6	10.713
Zero Mean Preprocessing					
SDM: 80% [32/44]	900	100.0	0.0	85.4	2.873
SDM: 85% [36/44]	700	**100.0**	**0.0**	**87.5**	**4.238**
SDM: 90% [42/44]	800	100.0	0.0	85.4	3.909
SDM: 95% [43/44]	800	100.0	0.0	85.4	3.909
Zero Mean and Unit Norm Preprocessing					
SDM: 80% [31/44]	750	**100.0**	**0.0**	**87.5**	**6.569**
SDM: 85% [34/44]	700	100.0	0.0	85.4	7.299
SDM: 90% [37/44]	950	100.0	0.0	81.2	6.993
SDM: 95% [40/44]	500	100.0	0.0	87.5	8.806

Table 19: SDM Transformation, 4-NN Runs

Flat Data w/ param = 1.0	PNN σ ×1000	Training		Testing	
		% TP	% FP	% TP	% FP
No Preprocessing					
SDM: 80% [29/44]	350	100.0	0.0	79.2	2.826
SDM: 85% [32/44]	650	**100.0**	**0.0**	**83.3**	**3.014**
SDM: 90% [36/44]	400	100.0	0.0	79.2	3.132
SDM: 95% [39/44]	600	100.0	0.0	81.2	3.838
Unit Norm Preprocessing					
SDM: 80% [32/44]	100	100.0	0.0	83.3	5.675
SDM: 85% [35/44]	100	100.0	0.0	79.2	4.474
SDM: 90% [38/44]	100	100.0	0.0	83.3	5.510
SDM: 95% [41/44]	100	**100.0**	**0.0**	**83.3**	**5.086**
Zero Mean Preprocessing					
SDM: 80% [30/44]	500	100.0	0.0	72.9	3.555
SDM: 85% [33/44]	350	100.0	0.0	83.3	4.191
SDM: 90% [36/44]	550	100.0	0.0	72.9	3.179
SDM: 95% [40/44]	700	**100.0**	**0.0**	**83.3**	**2.896**

Table 20: SDM Transformation / UNKL Runs[41]

Flat Data w/ param = 1.0	PNN σ ×1000	Training		Testing	
		% TP	% FP	% TP	% FP
No Preprocessing					
SDM: 80% [28/44]	500	100.0	0.0	75.0	3.697
SDM: 85% [31/44]	550	100.0	0.0	79.2	3.626
SDM: 90% [35/44]	350	100.0	0.0	79.2	3.814
SDM: 95% [39/44]	400	**100.0**	**0.0**	**81.2**	**4.756**
Unit Norm Preprocessing					
SDM: 80% [32/44]	100	**100.0**	**0.0**	**89.6**	**6.569**
SDM: 85% [35/44]	50	100.0	0.0	89.6	6.616
SDM: 90% [38/44]	50	100.0	0.0	83.3	6.664
SDM: 95% [41/44]	100	100.0	0.0	83.3	7.158
Zero Mean Preprocessing					
SDM: 80% [32/44]	600	100.0	0.0	85.4	4.921
SDM: 85% [36/44]	600	**100.0**	**0.0**	**85.4**	**4.097**
SDM: 90% [42/44]	500	100.0	0.0	81.2	4.285
SDM: 95% [43/44]	500	100.0	0.0	81.2	4.309

Table 21: SDM Transformation, 4-NN / UNKL Runs[41]

9.5 VQ Algorithm Results

Codebooks of size 100 through 8000 were generated for the six VQ algorithms. The MSE per pattern is plotted versus the codebook size in Figure 61, and the number of vectors retained after outlier removal is also plotted versus the codebook size in the same Figure. Some general observations can be made.

It is evident that in terms of MSE, the C&SL algorithm works best by far. The FSCL algorithm is second best, and the others are all pretty much the same, with OLVQ1 and OLVQ1C being the worst. In the bottom of the figure though, we see that MSE has

[41] The matrix A_{SDM}^{-T} for Zero Mean and Zero Norm could not be computed since A_{SDM}^{T} was singular.

Flat Data w/ param = 1.0	PNN σ ×1000	Training		Testing	
		% TP	% FP	% TP	% FP
No Preprocessing					
Regular [9,11]	650	100.0	0.0	75.0	4.168
UNKL [9,11]	900	100.0	0.0	77.1	3.932
4-NN Outlier Removal [9,12]	800	**100.0**	**0.0**	**83.3**	**4.450**
UNKL, 4-NN [9,12]	1000	100.0	0.0	81.2	4.992
Unit Norm Preprocessing					
Regular [10,12]	150	100.0	0.0	83.3	4.144
UNKL [10,12]	100	100.0	0.0	83.3	4.827
4-NN Outlier Removal [10,12]	50	100.0	0.0	83.3	4.803
UNKL, 4-NN [10,12]	100	**100.0**	**0.0**	**85.4**	**6.428**

Table 22: Composite Transform Runs

little to do with the classification ability (nearest neighbor) of the codebook. Here, the LBG algorithm retains by far the greatest number of vectors, and the C&SL algorithm retains the fewest. This should indicate that the LBG algorithm will give the best classification results (which is the case). For reference, the number of vectors retained after deterministic 4-NN and 8-NN outlier removal is also shown on the graph. We see that all of the VQ algorithms retain more vectors than the deterministic algorithms.

The classification runs were performed with a 12:6:1 neural network, where 1,000,000 iterations per simulation were run, and 10 simulations were run and averaged for every result reported. An extended mine region of 1 was used, and the data was balanced. All 11 frequencies and 4 S parameters were used in the simulations, giving an input vector size of 44.

The results from the simulations are shown in Table 28. The best results for each category are tabulated in Table 27. Here we see that the best results are obtained for the outlier removed data as opposed to the centroid data. This is different from the PNN and nearest neighbor classifiers, and indicates that the NN is able to generate more complex decision boundaries.

We see that only the LBG algorithm outperforms the 4-NN outlier removal, and even it does not give much better results (the true positive rate is only fractionally better, and the false positive rate is worse). This is regrettable, but it shows us that the VQ methods do not improve performance. It does enforce our belief that the 4-NN outlier removal method is a remarkably good method for its simplicity, however.

Flat Data w/ param = 1.0	Table #	Training		Testing	
		% TP	% FP	% TP	% FP
No Preprocessing					
KL	13	100.0	0.0	87.5	3.414
Composite	22	100.0	0.0	85.4	6.428
SDM	20	100.0	0.0	83.3	3.014
KL+	14	100.0	0.0	81.2	4.733
ESM	7	100.0	0.0	79.2	2.355
Benchmark	5	100.0	0.0	77.1	4.850

Table 23: Summary: No Preprocessing

Flat Data w/ param = 1.0	Table #	Training		Testing	
		% TP	% FP	% TP	% FP
Unit Norm Preprocessing					
KL	13	100.0	0.0	91.7	5.062
SDM	19	100.0	0.0	91.7	10.313
KL+	17	100.0	0.0	87.5	4.521
ESM	9	100.0	0.0	87.5	5.086
Benchmark	4	100.0	0.0	85.4	5.604

Table 24: Summary: Unit Norm Preprocessing

Flat Data w/ param = 1.0	Table #	Training		Testing	
		% TP	% FP	% TP	% FP
Zero Mean Preprocessing					
SDM	18	100.0	0.0	91.7	3.249
KL+	15	100.0	0.0	89.6	5.910
KL	11	100.0	0.0	87.5	5.910
ESM	7	100.0	0.0	79.2	1.648
Background	4	100.0	0.0	77.1	3.508

Table 25: Summary: Zero Mean Preprocessing

Flat Data w/ param = 1.0	Table #	Training		Testing	
		% TP	% FP	% TP	% FP
Zero Mean and Unit Norm Preprocessing					
KL+	17	100.0	0.0	89.6	6.664
KL	11	100.0	0.0	89.6	6.899
ESM	9	100.0	0.0	87.5	5.416
SDM	19	100.0	0.0	87.5	6.569
Benchmark	4	100.0	0.0	83.3	5.580

Table 26: Summary: Zero Mean and Unit Norm Preprocessing

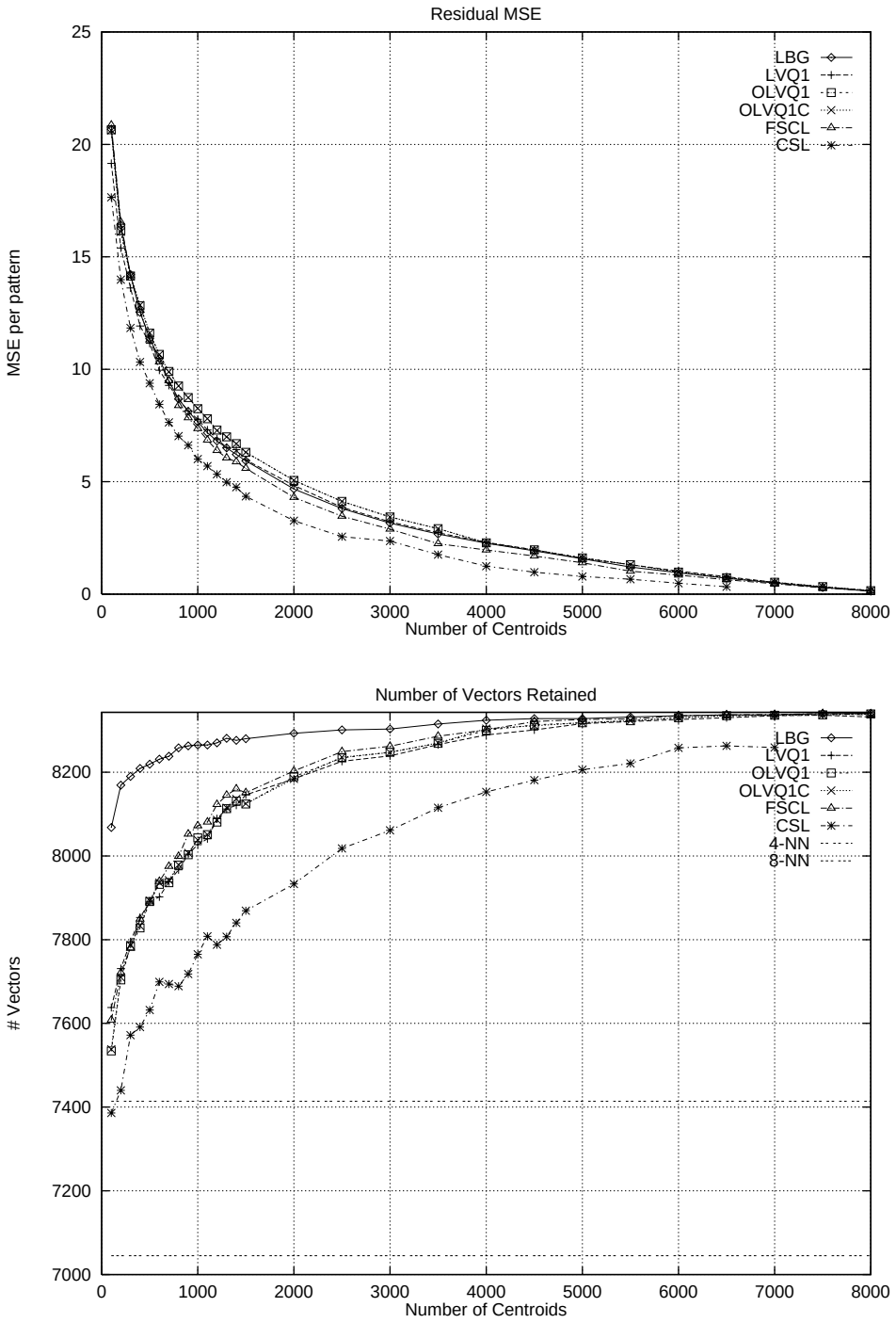


Figure 61: Codebook MSE/pattern and Number of Training Vectors Retained after Codebook Construction

Flat Data w/ param = 1.0	Training		Testing	
	% TP	% FP	% TP	% FP
VQ Algorithm				
LBG	99.8	4.706	97.7	8.776
4-NN	99.1	3.341	97.6	7.977
Benchmark	99.4	3.952	96.9	8.398
LVQ1	99.7	4.169	96.9	8.654
8-NN	99.5	4.535	96.9	9.831
FSCL	99.2	4.834	96.5	9.118
OLVQ1	99.5	3.911	95.8	8.089
OLVQ1C	98.6	3.476	95.2	7.617
C&SL	99.5	3.229	95.0	6.574

Table 27: Best VQ Algorithm Results

Flat Data w/ param = 1.0	Training		Testing	
	% TP	% FP	% TP	% FP
Benchmark Run				
Benchmark	99.4	3.952	96.9	8.398
4-NN Outlier Removal				
4-NN	**99.1**	**3.341**	**97.6**	**7.977**
8-NN Outlier Removal				
8-NN	99.5	4.535	96.9	9.831
LBG Runs (centroids)				
1000	95.3	6.555	88.1	8.799
1500	98.5	6.282	89.4	9.047
2000	98.0	6.237	91.5	9.399
2500	99.1	4.684	92.7	8.568
3000	99.2	6.656	94.0	9.424
3500	**99.5**	**6.526**	**95.4**	**9.886**
LBG Runs (outliers removed)				
1000	99.8	4.006	96.7	8.350
1500	99.8	4.307	96.7	7.953
2000	99.7	4.282	96.3	8.176
2500	99.7	4.713	96.7	8.741
3000	**99.8**	**4.706**	**97.7**	**8.776**
3500	99.7	5.016	97.1	9.015
4000	99.8	3.539	95.2	7.711
LVQ1 Runs (centroids)				
1000	93.0	5.786	84.8	8.720
1500	97.4	4.770	90.0	8.301
2000	98.0	5.077	91.9	8.639
2500	**99.4**	**5.704**	**92.7**	**9.566**
3000	98.6	5.060	91.9	8.593
LVQ1 Runs (outliers removed)				
1000	93.5	5.591	89.6	9.073
1500	97.6	3.238	92.9	7.455
2000	98.5	3.245	94.4	7.138
2500	99.1	5.169	93.8	8.559
3000	**99.7**	**4.169**	**96.9**	**8.654**
OLVQ1 Runs (centroids)				
1000	94.1	7.115	87.5	9.971
1500	96.8	4.758	88.1	8.267
2000	97.1	6.338	90.2	9.435
2500	**99.1**	**4.831**	**92.3**	**8.398**
3000	98.8	4.576	91.0	7.888
OLVQ1 Runs (outliers removed)				
1000	92.3	3.267	88.7	6.584
1500	96.5	4.451	93.1	8.545
2000	98.2	3.834	95.2	7.858
2500	**99.5**	**3.911**	**95.8**	**8.089**
3000	99.7	4.240	94.2	8.051

Flat Data w/ param = 1.0	Training		Testing	
	% TP	% FP	% TP	% FP
OLVQ1C Runs (centroids)				
1000	77.0	6.631	71.2	9.675
1500	96.2	4.205	87.9	7.602
2000	97.6	4.869	90.8	8.068
2500	**99.2**	**5.648**	**91.9**	**9.255**
3000	98.8	4.887	90.8	8.342
OLVQ1C Runs (outliers removed)				
1000	77.3	6.194	73.1	8.849
1500	96.7	5.223	94.6	8.510
2000	98.6	3.476	95.2	7.617
2500	99.4	5.507	94.6	8.713
3000	99.4	5.404	95.2	8.294
FSCL Runs (centroids)				
1000	91.5	7.476	85.4	10.221
1500	97.1	5.817	89.4	8.260
2000	**98.6**	**4.637**	**93.1**	**7.529**
2500	99.2	4.212	90.2	7.820
3000	98.3	4.742	92.9	9.132
FSCL Runs (outliers removed)				
1000	93.9	2.910	89.8	6.144
1500	97.9	5.147	93.7	8.379
2000	98.5	5.455	95.2	9.322
2500	**99.2**	**4.834**	**96.5**	**9.118**
3000	99.7	4.587	94.8	8.793
C & SL Runs (centroids)				
1000	88.5	7.190	87.1	9.047
1500	97.4	4.873	91.2	7.765
2000	98.9	4.930	92.7	7.662
2500	97.1	4.566	91.2	7.877
3000	**99.4**	**4.711**	**94.4**	**8.041**
C & SL Runs (outliers removed)				
1000	92.9	5.544	86.0	7.271
1500	98.3	2.705	93.5	6.402
2000	99.1	3.419	94.4	6.830
2500	98.9	3.757	94.0	7.154
3000	**99.5**	**3.229**	**95.0**	**6.574**

Table 28: Various Algorithm VQ Results

10 Backpropagation Neural Network Performance

In this section, we summarize the performance achieved by using a backprop neural network for land mine detection. The effect of modifying some of the variable factors in this system are discussed. Some of the relevant result tables are given here and others are available in previous reports [11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21].

10.1 Network Architecture

During the course of this contract and its predecessor, U.S. Army Contract DAAK70-89-K-0001, several different types of network architecture was investigated. These include fully connected feedforward networks, partially connected or bottlenecked networks, and recurrent networks. The recurrent networks were found to be too computationally intensive during training to be very useful [30]. The bottlenecked networks were studied in an attempt to reduce the number of weights necessary in the network. However, any performance improvement seen using such a network was outweighed by the difficulty of designing and choosing the proper network configuration. Therefore, a fully connected feedforward neural network was considered the best choice for this work.

In addition to the interconnectivity, the size of the network was also extensively studied. It was found that a three layer (two hidden layers and one output layer) network gave the best performance. A two layer network had difficulty modeling the complex relationship between the input signals and the classification output, and a four layer network was unnecessarily complex. Through many simulations, the optimum number of neurons to use in each layer was also established. For the single output case, which was studied most, three layers consisting of eight neurons in the first hidden layer, three neurons in the second hidden layer, and one neuron in the output layer seemed to work very well. Varying the number of neurons by a small number in each layer showed only minor differences in the performance characteristics. Dr. Torrieri reported good results for a 12:6:1 configuration which was used to test the VQ and NN methods in the previous section. Therefore, a small range of network size was established as an acceptable network architecture. The 8:3:1 configuration consistently showed good results using the backprop neural network and is recommended for continued research. The number of inputs used will be discussed in the next section.

10.2 Input Data Types

Not only is there a large variety of available network architectures, but the choice of input signals is also very extensive. As described in Section 1.2, 88 different measurements were taken at each spatial location in the mine lane. This includes the magnitude and phase of four different S parameters at 11 separate frequencies.

Initially, all work was done with a spatial window ranging from 7×7 to 13×13.

These large input windows made it formidable to use more than one or two S parameters or frequencies in the input vector. For this reason and also to make the system implementation simpler, we eventually migrated to the 1×1 or minimal window. When using only one spatial point of data, a more complex mixture of S parameters and frequencies were usable.

Using the minimal window still gives us a possibility of 88 inputs. This is still too large for the methods used here. Therefore, simulations were conducted to determine the most vital parameters for mine detection. For a neural network of comparable size (similar number of weights), it was determined that only seven frequencies (880 MHz to 1120 MHz) was necessary to obtain satisfactory results. Using additional frequencies added very little to the performance, and using fewer frequencies started to significantly degrade the performance. In addition, including the phase information reduced the performance of the network.

From the above results, it can be concluded that a 28 component input vector consisting of the magnitude components of all four S parameters obtained at the seven frequencies between 880 MHz and 1120 MHz were sufficient to perform optimum mine detection.

10.3 Effects of Preprocessing

The research conducted on preprocessing the input data before using a neural network proved to be vital for optimum performance. As described in Section 4.1.2, the raw data originally included some unnatural peculiarities in the form of an underlying "ramp" and also some "jumps." The removal of these characteristics from the data was able to dramatically increase the performance of the neural network mine detector. Furthermore, the deterministic removal of outliers from the training data set (Section 6) and balancing the mine data to background data ratio proved to be very beneficial. These techniques are recommended in all future studies and implementations of this system. Both the four nearest neighbor and eight nearest neighbor outlier removal methods work very well, and research should continue with both methods until one is determined to be significantly better than the other.

Another class of preprocessing methods which proved to be very promising is the use of transform methods (Section 7). All of the transform methods studied reduced the input vector and network sizes and improved performance over the benchmark case. Both the Karhunen Loeve transform and the Eigenspace Separation Method showed good results. The KL transform with 28 inputs achieved the best peak performance of 100% mine detection with only 9.1% false positives. The ESM which only used 16 inputs was able to produce better average performance than the corresponding 16 input KL transform. In addition, using ESM, a reduced size network of 6:3:1 was able to achieve comparable performance [46]. Some of these results are given in Table 29. All of these results are given with an extended mine region of one, and the 4-nearest neighbor outlier removal method was used for all

of them. Some of the corresponding results obtained using the 8-nearest neighbor outlier removal method are given in Table 30. The use of these transform methods and possibly others are a very promising area of future research in neural network mine detection.

10.4 Detection Rate Tradeoffs

When evaluating the performance of a classification system, there are two important criteria: the true positive rate and the false positive rate. In addition, these two rates are not independent of each other. In fact, there is a strong correlation between the two measures and a tradeoff must be made when maximizing the performance of the system. During most of the research conducted under this contract, emphasis was placed on maximizing the mine detection (true positive, TP) rate. All the results reported use this method as described in Section 1.4.4. In many cases, perfect detection or 100% detection was achieved in the test region. Although the best possible detection rate is achieved, there are cases when the false positive (FP) rate may be undesirably high. Therefore, the question arises whether some of the detection performance can be sacrificed for a better false positive rate.

As an example, we review the case of 28 inputs using the four nearest neighbor outlier removal method with data balancing. Using the maximum detection criteria, the peak performance achieved was 100.0% TP with 11.4% FP. An averaged measure over 20 different initial conditions resulted in 96.5% TP and 12.2% FP. However, if we use a minimum detection rate of 95% as the performance criteria[42], the result change slightly. The best performance now becomes 95.8% TP with 8.1% FP, and the average becomes 95.8% TP and 10.2% FP. For both the best and averaged result, the mine detection rate and the false positive rate has decreased. Furthermore, if the minimum detection rate is reduced to 90%, the best performance becomes 91.7% TP with 6.3% FP, and the averaged result becomes 92.0% TP and 8.3% FP. Again, there is further reduction in both the detection rate and the false positive rate. This demonstrates that the tradeoff between the true positive detection rate and the false positive rate is very complex. These results are tabulated in Table 31. At this time, it has not been determined what kind of figures are desirable or acceptable. This may be best left up to the user of the system.

Other statistical analyses, such as Receiver Operating Characteristic (ROC) analysis, can be made to determine the tradeoffs and overall performance characteristic of the classification system [26, 32, 44]. This would be an area of great significance in future research of the neural network mine detection system.

10.5 Multiple Neural Network Outputs

The majority of the work represented in this report uses a single neural network output (Section 5.1). However, the latest research shows that using multiple outputs (6 outputs,

[42] For this new performance criteria, all neural network results with at least a 95% TP rate will be examined and the one with the lowest FP rate will be chosen as the best result.

one for each mine type and one for background) may also be useful. Preliminary results show that a peak detection rate of 97.9% with a false positive rate of 11.2% was achieved. The average detection rate over 10 different initial conditions was 93.3% with a false positive rate of 9.7%. Furthermore, using the composite transform with 13 inputs (Section 7.3), a 97.9% detection rate was possible with a false positive rate of 8.3%. The average detection rate was 94.6% with a false positive rate of 9.9%. In order to compare neural networks with a comparable number of weights, a 15:3:1 network architecture was used with the composite algorithm, and an 8:3:1 architecture for the non-transformed input case. These results are very promising. Therefore, future work involving multiple output neural network mine detectors including special preprocessing procedures to optimize the discrimination between different mine types is highly recommended.

Transform Method	Averaged Results (20sims)				Peak Performance			
	Training		Testing		Training		Testing	
	% TP	% FP	% TP	% FP	% TP	% FP	% TP	% FP
None	99.2	10.4	96.5	12.2	100.0	8.7	100.0	11.4
KL [28/28]	99.6	8.9	97.4	11.1	100.0	5.8	100.0	9.1
KL [16/28]	98.6	11.4	96.7	12.8	100.0	10.6	100.0	11.4
ESM	99.3	11.1	97.5	12.2	98.5	10.9	100.0	11.7
ESM/UNKL	99.3	11.7	96.9	13.0	100.0	17.7	100.0	15.6
Composite	97.0	12.7	96.6	14.0	98.5	12.1	100.0	12.4
Composite/UNKL	97.3	16.1	97.1	15.6	93.9	11.9	100.0	10.3

Table 29: Mine detection performance comparing transform preprocessing methods using the 4-nearest neighbor outlier removal method.

Transform Method	Averaged Results (20sims)				Peak Performance			
	Training		Testing		Training		Testing	
	% TP	% FP	% TP	% FP	% TP	% FP	% TP	% FP
None	99.4	10.5	97.7	12.9	100.0	8.4	100.0	9.8
KL [28/28]	99.5	9.4	98.0	12.7	100.0	11.2	100.0	11.6
KL [16/28]	99.0	10.1	97.3	13.0	98.5	8.1	100.0	9.2
ESM	98.8	11.4	97.8	13.2	100.0	8.2	100.0	9.3
ESM/UNKL	98.6	11.3	97.6	12.5	100.0	8.8	100.0	11.0

Table 30: Mine detection performance comparing transform preprocessing methods using the 8-nearest neighbor outlier removal method.

Performance Criteria	Averaged Results (20sims)				Peak Performance			
	Training		Testing		Training		Testing	
	% TP	% FP	% TP	% FP	% TP	% FP	% TP	% FP
Maximum Detection	99.2	10.4	96.5	12.2	100.0	8.7	100.0	11.4
95% Detection	99.8	8.4	95.8	10.2	100.0	8.8	95.8	8.0
90% Detection	99.2	7.2	92.0	8.3	98.5	6.8	91.7	6.3

Table 31: Mine detection and false alarm tradeoffs.

A Target Locations

This appendix tabulates the locations of all the targets in the "SAS6" data set. They are organized according to mine type, and sorted by row. The locations are in mine lane grid coordinates, and indicate the center of the mine.

Mine Type 1			
Row	Col	Rot	Num
8.0	20.2	62	0
73.3	9.0	101	1
105.3	9.6	61	2
125.7	12.9	145	3
179.5	16.6	50	4
212.8	24.5	9	5
245.9	11.8	0	6
272.5	5.5	135	7
289.8	8.1	169	8
314.0	23.3	71	9
387.3	4.5	40	10
393.0	19.9	160	11
404.3	8.9	67	12
441.1	6.1	65	13
450.0	21.2	31	14

Mine Type 2			
Row	Col	Rot	Num
18.3	11.0	0	15
29.0	19.8	0	16
59.0	8.6	0	17
78.2	23.6	0	18
85.0	4.3	0	19
116.0	20.0	0	20
137.0	24.0	0	21
154.2	6.7	0	22
168.0	24.5	0	23
184.9	6.5	0	24
225.5	10.4	0	25
232.9	23.7	0	26
252.0	24.3	0	27
261.4	3.8	0	28
282.1	23.2	0	29
307.6	15.5	0	30
324.6	4.2	0	31
329.2	24.6	0	32
356.8	10.5	0	33
376.9	19.6	0	34
431.0	2.2	0	35
439.2	18.9	0	36
454.2	2.1	0	37
461.3	25.1	0	38
476.1	8.6	0	39
491.2	10.5	0	40
492.5	24.3	0	41
503.4	2.2	0	42
505.9	17.6	0	43
515.0	24.5	0	44

Mine Type 3			
Row	Col	Rot	Num
11.7	3.3	0	45
49.5	6.5	0	46
62.9	20.5	0	47
143.1	7.6	0	48
169.1	3.6	0	49
199.6	24.7	0	50
233.1	5.7	0	51
258.6	12.6	0	52
331.1	11.6	0	53
349.2	6.1	0	54
430.0	23.7	0	55
463.5	12.4	0	56

Mine Type 4			
Row	Col	Rot	Num
38.1	9.4	87	57
41.1	19.7	2	58
89.9	19.0	0	59
159.0	19.3	42	60
197.0	10.8	116	61
216.2	6.0	163	62
264.0	24.5	14	63
297.3	19.3	153	64
309.6	5.0	105	65
343.9	20.6	22	66
364.5	22.8	95	67
364.8	9.2	25	68
413.6	19.0	90	69
420.5	5.0	71	70
475.0	21.6	94	71

Mine Type 5			
Row	Col	Rot	Num
18.3	21.0	45	72
24.6	6.0	35	73
27.0	26.0	301	74
54.1	19.0	122	75
79.8	14.5	128	76
94.8	6.4	270	77
100.6	22.5	90	78
117.6	5.0	170	79
125.2	18.2	150	80
133.5	6.5	230	81
140.0	14.0	330	82
142.5	26.0	350	83
163.7	10.8	182	84
173.3	10.7	170	85
179.5	20.7	0	86
202.7	21.2	90	87
204.9	4.5	204	88
213.7	15.4	293	89
234.6	14.0	180	90
252.5	9.2	254	91
273.5	15.3	220	92
278.8	18.5	321	93
299.5	7.5	299	94
315.0	5.8	263	95
322.0	16.1	9	96
335.8	11.3	225	97
354.2	21.3	178	98
364.1	4.3	244	99
380.4	8.0	241	100
384.7	21.0	70	101
401.9	15.5	218	102
403.5	22.7	232	103
422.6	19.2	80	104
431.3	13.0	178	105
453.7	15.0	193	106
461.7	4.3	327	107
465.8	19.1	31	108
483.6	17.2	54	109
485.8	3.7	6	110
495.3	16.5	0	111
513.9	7.4	15	112
519.4	16.5	173	113

The following plot is a "dump" of the mine lane. All 527 rows and 27 columns are displayed. Each row is identified on the left as being either in the "train"ing section or in the "test"ing section. Points in the mine lane which correspond with a grid square which has some section of a mine underneath it are labeled with a number from 1 to 5, corresponding to the type of the mine. Points corresponding to an extended mine region of 1 are noted with '–', and points corresponding to an extended mine region of 2 are noted with '='. Purely background points are noted with '.'.

```
  0: train ..........................
  1: train ..........................
  2: train ..........................
  3: train ...............========...
  4: train ...............==------=...
  5: train ...............=--1111-=...
  6: train ...............=-11111-=...
  7: train ...............=-11111-=...
  8: train ======.........=-11111-=...
  9: train -----=.........=--1111-=...
 10: train -333-=.........==------=...
 11: train -333-=..........========...
 12: train -333-=...................
 13: train -----=...................
 14: train ======.=======...======....
 15: train ......=======..=----==...
 16: train ......=--222--=..=-55--=...
 17: train ......=-22222-=..=-555-=...
 18: train ......=-22222-=..=-555-=...
 19: train ......=--222--=..=--55-=...
 20: train ..==========--==..=======...
 21: train ..=----------....======...
 22: train ..=-55--=................
 23: train ..=-555-=.............====
 24: train ..=-555-=.......========---
 25: train ..=--55-=......==-----=--55
 26: train ..==----=......=--222-=-555
 27: train ...======......=-2222-=-55-
 28: train ..............=-2222-=----
 29: train ..............=-2222-=====
 30: train ..............=--222-=....
 31: train ..............==-----=....
 32: train ..............=======....
 33: train ....=========............
 34: train ....=-------=............
 35: train ....=-44444-=............
 36: train ....=-44444-=..=========...
 37: train ....=-44444-=..=-------=...
 38: train ....=-44444-=..=-44444-=...
 39: train ....=-44444-=..=-44444-=...
 40: train ....=-------=..=-44444-=...
 41: train ....=========..=-44444-=...
 42: train ..............=-44444-=...
 43: train ..............=-------=...
 44: train ..............=========...
 45: train ...======................
 46: train ..==-----==..............
 47: train ..=--33--=...............
 48: train ..=-3333-=...............
 49: train ..=-3333-=...............
 50: train ..=--33--=......======.....
 51: train ..==------==.....==----=.....
 52: train ...======......=--55-=.....
 53: train ..............=-555-=.....
 54: train .....======....=-555-=.....
 55: train ....==-----==...=-----=.....
 56: train ....=--22--=...=======.....
 57: train ....=-2222-=..............
 58: train ....=-2222-=..............
 59: train ....=-2222-=....========...
 60: train ....=--22--=....=------=...
 61: train ....==-----==....=-3333-=...
 62: train .....======.....=-3333-=...
 63: train ..............=-3333-=...
 64: train ..............=------=...
 65: train ..............========...
 66: train ..........................
 67: train ..........................
 68: train .....=======..............
 69: train ....==------==............
 70: train ....=--111--=............
 71: train ....=-11111-=............
 72: train ....=-11111-=............
 73: train ....=-11111-=............
 74: train ....=-11111-=......========
 75: train ....=-------=......=------=
 76: train ....===============.=-2222-=
 77: train ..........==-----=.=-2222-=
 78: train ..........=--555-=.=-2222-=
 79: train ..........=-5555-=.=--222-=
 80: train .======...=-555--=.=------=
 81: train ==-----==..------==..=======
 82: train =--22--=..=======..........
 83: train =-2222-=..................
 84: train =-2222-=..................
 85: train =-2222-=......=========....
 86: train =--22--=......=-------=....
 87: train ==-----==......=-44444-=....
 88: train .======.......=-44444-=....
 89: train ..............=-44444-=....
 90: train ...======.....=-44444-=....
 91: train ...=----=.....=-44444-=....
 92: train ...=-55-=.....=-------=....
 93: train ...=-55-=.....=========....
 94: train ...=-55-=..................
 95: train ...=-55-=..................
 96: train ...=----=.........======..
 97: train ...======..........=----=..
 98: train ....................=-55-=..
 99: train ....................=-55-=..
100: train ......======.......=-55-=..
101: train .....==------==......=-55-=..
102: train .....=--11--==.....=----=..
103: train .....=-1111--=.....======..
104: train .....=-11111-=.............
105: train .....=-11111-=.............
106: train .....=-1111--=.............
107: train .....=------==.............
108: train .....========..............
109: train ..........................
110: test  ..........................
111: test  ................=======....
112: test  ..............==------==...
113: test  ..............=--222--=...
```

```
114: test  .=======.......=-22222-=...
115: test  .=-----=.......=-22222-=...
116: test  .=-555-=.......=-22222-=...
117: test  .=-555-=.......=--222--=...
118: test  .=-----=.......==-----==...
119: test  .=======........=======....
120: test  ..........................
121: test  ........============......
122: test  ........=----------=......
123: test  ........=-1111---55-=......
124: test  ........=-11111-555-=......
125: test  ........=-11111-555-=......
126: test  ........=-11111-----=......
127: test  ........=--111--=====......
128: test  ........==------==.........
129: test  ...============...........
130: test  ...=----=.................
131: test  ...=-55-=.................
132: test  ...=-55-=..........=======
133: test  ...=-55-=..........==-----=
134: test  ...=-55-=..........=--222--
135: test  ...=----=.........=-22222-
136: test  ...======..======..=-22222-
137: test  ..........==----=..=-22222-
138: test  ..........=--55-=..=--222--
139: test  ....=======-555-=..==-----=
140: test  ....=-----=-55--=...===----
141: test  ....=-333-=----==.....=-555
142: test  ....=-333-======......=-555
143: test  ....=-333-=...........=----
144: test  ....=-----=...........=====
145: test  ....=======..............
146: test  ..........................
147: test  ..........................
148: test  ..........................
149: test  ..........................
150: test  ..========................
151: test  ..=------=................
152: test  ..=-2222-=................
153: test  ..=-2222-=......======.....
154: test  ..=-2222-=.....==-----==....
155: test  ..=--222-=....==--44--==...
156: test  ..=------=....=--4444--=...
157: test  ...=======....=-444444-=...
158: test  ..............=-444444-=...
159: test  ..............=-444444-=...
160: test  ......=========--4444--=...
161: test  ......=------===--44--==...
162: test  ......=-5555-=.==-----==....
163: test  ......=-5555-=..===========
164: test  ......=------=......==----=
165: test  ==============......=--22--
166: test  =-----=.............=-2222-
167: test  =-333-=.............=-2222-
168: test  =-333-=.............=-2222-
169: test  =-333-=.======......=--22--
170: test  =---------=-----=......==----=
171: test  =========--55-=.......======
172: test  ......=-5555-=.............
173: test  ......=-5555-=.............
174: test  ......=------=.............
175: test  ......==============.......
176: test  ............=------=====...
177: test  ............=-1111-----=...
178: test  ............=-11115555-=...
179: test  ............=-11115555-=...
180: test  ...======...=-1111-----=...
181: test  ..=------==..=------=====...
182: test  ..=--22--=..========.......
183: test  ..=-2222-=.................
184: test  ..=-2222-=.................
185: test  ..=-2222-=.................
186: test  ..=--22--=.................
187: test  ..==----==.................
188: test  ...======..................
189: test  ..........................
190: test  ..........................
191: test  ..........................
192: test  ......========............
193: test  ......=------==............
194: test  .....==-4444--=............
195: test  .....=--44444-=......======
196: test  .....=-444444-=......=----=
197: test  .....=-444444-=......=-33--
198: test  .....=--4444--=...====-333-
199: test  .....==---44-==...=----333-
200: test  ......===----=....=-55-33--
201: test  =======.======....=-55-----=
202: test  =------==.........=-55-====
203: test  =-555--=..........=-55-=...
204: test  =-5555-=..........=----=...
205: test  =--555-=..........======...
206: test  ==-----=...................
207: test  .=======...................
208: test  ....................=======
209: test  ............======..=------
210: test  ............=----=..=-1111-
211: test  ..========.==-55-=..=-1111-
212: test  .==------=.=--55-=..=-1111-
213: test  .=--4444-=.=-555-=..=-1111-
214: test  .=-44444-=.=-555-=..=-1111-
215: test  .=-44444-=.=------=..=------
216: test  .=-44444-=.=======..=======
217: test  .=-44444-=.................
218: test  .=-------=.................
219: test  .=========.................
220: train ..........................
221: train ......========............
222: train ......=------=............
223: train ......=-2222-=............
224: train ......=-2222-=............
225: train ......=-2222-=............
226: train ......=-2222-=............
227: train ......=------=............
228: train ......========......=======
229: train ..=======..........==-----=
230: train ..=-----=..........=--222-=
231: train ..=-333-=.=======..=-2222-=
232: train ..=-333-=.=------=..=-2222-=
233: train ..=-333-=.=-555-=..=-2222-=
234: train ..=-----=.=-555-=..=--22--=
235: train ..=======.=------=..==-----==
236: train ..........=======...======.
237: train ..........................
238: train ..........................
239: train ..........................
240: train ..........................
241: train .......========............
242: train .......=------==...........
243: train .......=-1111--=...........
244: train .......=-11111-=...........
245: train .......=-11111-=...........
246: train .......=-11111-=...........
247: train .......=--111--=.....======
248: train .....====------==....==----=
249: train .....=-----====.....=--22--
250: train .....=-555-=........=-2222-
251: train .....=-555-=........=-2222-
252: train .....=--55-=........=-2222-
253: train .....==-55-=........=--22--
254: train ......=----====.....==----=
255: train ......====----==.....======
256: train ........=--33--=...........
257: train =======.=-3333-=...........
258: train ------=.=-3333-=...........
259: train -2222-=.=--33--=....=======
```

```
260: train -2222-=.=-----==....=------
261: train -2222-=..======....==-44444
262: train --222-=...........=--44444
263: train =-----=...........=-444444
264: train =======...........=-44444-
265: train .................=-44444-
266: train .................=-------
267: train ..======.........========
268: train .==-----==...............
269: train ==--11--==.======........
270: train =--1111--=.=----==.......
271: train =-111111-=.=-55--=.......
272: train =-111111-=.=-555-=.......
273: train =--1111--=.=-555-=.......
274: train ==--11--==.=--55-=.......
275: train .==-----=..==---------....
276: train ..======....====-----=....
277: train ............=--555-=====.
278: train ............=-5555-----==
279: train ............=-555---22--=
280: train ............=-----2222-=
281: train ............======-2222-=
282: train ................=-2222-=
283: train ................=-222--=
284: train ................=-----==
285: train ....========.......=======.
286: train ...==------=.............
287: train ...=--1111-=.............
288: train ...=-11111-=.............
289: train ...=-11111-=.............
290: train ...=-11111-=.............
291: train ...=--111--=.............
292: train ...==------==...========....
293: train ....=======...==------=....
294: train ............=--4444-==...
295: train ....======....=-44444--=...
296: train ....=----=....=-444444-=...
297: train ....=-55-=....=-444444-=...
298: train ....=-55-=....=--4444--=...
299: train ....=-55-=....==-44---==...
300: train ....=-55-=.....=----===....
301: train ....=----=.....======......
302: train ....======..............
303: train ..........========.......
304: train =======....=------=.......
305: train =-----===..=-2222-=.......
306: train =-444---=..=-2222-=.......
307: train =-44444-=..=-2222-=.......
308: train =-44444-=..=-2222-=.......
309: train =-44444-=..=---------------
310: train =-44444-=..===============
311: train =--4444-=.........=--1111-=
312: train ==------=.........=-11111-=
313: train .==-55--=.........=-11111-=
314: train ..=-555-=.........=-11111-=
315: train ..=-555-=.........=--1111-=
316: train ..=-----=.........=-------=
317: train ..=======..........========
318: test  ............=======........
319: test  ............=-----=........
320: test  =======.....=-555-=........
321: test  =-----==....=-555-=........
322: test  =-222--=....=-555-=........
323: test  =-2222-=....=-----=........
324: test  =-2222-=....=======........
325: test  =-2222-=............=======
326: test  =------=............=------
327: test  ===============.....=-2222-
328: test  ........=-----=.....=-2222-
329: test  ........=-333-=.....=-2222-
330: test  ........=-333-=.....=--222-
331: test  ........=-333-=.....==-----
332: test  .......==-----=......======
333: test  .......=-----==............
334: test  .......=-555-=............
335: test  .......=-555-=............
336: test  .......=--55-=............
337: test  .......==----=............
338: test  ........======..======.....
339: test  ................=---===...
340: test  ................=-44---==..
341: test  ...............==-4444--=..
342: test  ...............=--44444-=..
343: test  ...............=-444444-=..
344: test  ...............=-444444-=..
345: test  ..=======......=--4444--=..
346: test  ..=-----=......==------==..
347: test  ..=-333-=.......========...
348: test  ..=-333-=..................
349: test  ..=-333-=..................
350: test  ..=-----=.........======..
351: test  ..=======........===----=..
352: test  ......========...=---55-=..
353: test  ......=------=...=-5555-=..
354: test  ......=-2222-=...=-5555-=..
355: test  ......=-2222-=...=------=..
356: test  ......=-2222-=...========..
357: test  ......=-2222-=.............
358: test  ......=------=.............
359: test  .....=========....=======..
360: test  ======-------.....=-----===
361: test  =------44---==....=-444---=
362: test  =-555--4444--=....=-44444-=
363: test  =-555-444444-=....=-44444-=
364: test  =--55-444444-=....=-44444-=
365: test  ==-----44444--=....=-44444-=
366: test  .====--4444-==....=---444-=
367: test  ....==------=.....=========
368: test  .....========.......=======
369: test  ............................
370: test  ............................
371: test  ............................
372: test  ...............=======....
373: test  ...............==-----=....
374: test  ...............=--222-=....
375: test  ...............=-2222-=....
376: test  ....======.....=-2222-=....
377: test  ....=-----==....=-2222-=....
378: test  ....=-55--=....=--22--=....
379: test  ....=-555-=....==-----==....
380: test  ....=-555-=.....=======....
381: test  ....=--55-=......=----==...
382: test  .=====-----=......=-55--=...
383: test  ==-----=====......=-555-=...
384: test  ==--11--=.........=-555-=...
385: test  =-1111-=.........=--55-=....
386: test  =-1111-=.........=======...
387: test  =-1111-=..........======...
388: test  =-1111-=........=======....
389: test  =-------=.......==-----==...
390: test  ========.......=--111--=...
391: test  ...............=-11111-=...
392: test  ...............=-11111-=...
393: test  ...............=-11111-=...
394: test  ...............=--111--=...
395: test  ...............==-----==...
396: test  ................=======....
397: test  ............................
398: test  ...........=======.........
399: test  .....========-----========..
400: test  ....==------=-555--=------==.
401: test  ....=--111---5555-===-55--=.
402: test  ....=-11111---555-===-555-=.
403: test  ....=-11111-=-------==-555-=.
404: test  ....=-11111-=========--55-=.
405: test  ....=-11111-=......=======.
```

```
406: test  ....=-------=.......======.
407: test  ....=========.............
408: test  ........................
409: test  ..............=========....
410: test  ..............=-------=....
411: test  ..............=-44444-=....
412: test  ..............=-44444-=....
413: test  ..............=-44444-=....
414: test  ..............=-44444-=....
415: test  ..======......=-44444-=....
416: test  ===-----==.....=-------=....
417: test  =---44--=.....=========....
418: test  =-44444-=......=======.....
419: test  =-44444-=......=-----=.....
420: test  =-44444-=......=-555-=.....
421: test  =-44444-=......=-555-=.....
422: test  =--44---=......=--55-=.....
423: test  ==------===......==-55-=.....
424: test  .======.........=----=.....
425: test  ................======.....
426: train =====..............=======
427: train -----==.............=-----=
428: train 222--=...=======....=-333-=
429: train 2222-=...=-----=....=-333-=
430: train 2222-=...=-555-=....=-333-=
431: train 2222-=...=-555-=....=-----=
432: train 222--=...=-----=....=======
433: train ----==...=======...........
434: train =====.......................
435: train ..............========.....
436: train ..=======.....=------==....
437: train .==------==....=-2222--=....
438: train .=--111--=....=-22222-=....
439: train .=-11111-=....=-22222-=....
440: train .=-11111-=....=--222--=....
441: train .=-11111-=....=======.....
442: train .=--1111-=.....=======.....
443: train .==------=.................
444: train ..========................
445: train ................========..
446: train ..............==------=..
447: train ..............=--1111-=..
448: train ..............=-11111-=..
449: train ..............=-11111-=..
450: train ======.....======-11111-=..
451: train -----=.....=------1111-=..
452: train 2222-=.....=-555-=------=..
453: train 2222-=.....=-555-========..
454: train 2222-=.....=-----=.........
455: train 222--=.....=======.........
456: train -----==....................
457: train =====..............======
458: train ========............==-----
459: train =------=.======.....=--2222
460: train =-5555-===-----==....=-22222
461: train =-5555-==--33--=....=-22222
462: train =-55---==-3333-======--222-
463: train =---------3333-=-----=-----
464: train ======..=--33--=-555-======
465: train ........==-------555-=.....
466: train .........========--55-=.....
467: train ..............==-----=.....
468: train ..............======.....
469: train .......................
470: train ..............=========.
471: train .....======......=-------=.
472: train ....=-----==.....=-44444-=.
473: train ....=--22--=....==-44444-=.
474: train ....=-2222-=....=--44444-=.
475: train ....=-2222-=....=-444444-=.
476: train ....=-2222-=....=-44444--=.
477: train ....=--222-=....=--------==.
478: train ....=------=....=========..
479: train .....=======.======........
480: train ............=----==.......
481: train ............=-55--=.......
482: train =======......=-555-=.......
483: train ------=......=-555-=.......
484: train -5555-=......=--55-=.......
485: train -5555-=......==----=.......
486: train ---55-=.......======.......
487: train ==-----------.............
488: train .======------=.....=======
489: train ......=-2222-=......=------
490: train ......=-2222-=......=-2222-
491: train ......=-2222-=......=-2222-
492: train ......=-2222-========-2222-
493: train ......=-----------==-2222-
494: train ......=======-5555-==------
495: train .............=-5555-========
496: train ............=------=.......
497: train ............========.......
498: train .......................
499: train ======.....................
500: train -----=.....................
501: train 2222-=........=======......
502: train 2222-=.......==-----=......
503: train 2222-=.......=--222-=......
504: train 222--=.......=-2222-=......
505: train ----==.......=-2222-=......
506: train =====........=-2222-=......
507: train ..............=--22--=......
508: train ..............==----==......
509: train ..............======........
510: train ...=======...........======
511: train ...=-----==.........==-----=
512: train ...=-555--=.........=--22--
513: train ...=-5555-=.........=-2222-
514: train ...=--555-=.........=-2222-
515: train ...==-----=.........=-2222-
516: train ....=======.=========--22--
517: train ............=------===-----=
518: train ............=-5555-=.======
519: train ............=-5555-=.......
520: train ............=------=.......
521: train ............========.......
522: train .......................
523: train .......................
524: train .......................
525: train .......................
526: train .......................
```

B Derivation of the SDM Transformation

This section essentially duplicates the analysis performed in [24, p 33], but is included here for completeness, and to introduce the notation necessary for the next section. We define:

Σ_- is the covariance matrix calculated using only the vectors in the mine lane belonging to the background class.

Σ_+ is the covariance matrix calculated using only the vectors in the mine lane belonging to mine classes 1-5.

The first step of the transform is to whiten Σ_- by:

$$Y = \theta^{-\frac{1}{2}}\Phi^T X$$

where Φ contains, as its columns, the orthonormal eigenvectors of Σ_-, and where θ is a diagonal matrix whose entries are the corresponding eigenvalues of Σ_-. i.e.

$$\Sigma_-\Phi = \Phi\theta \qquad \text{and} \qquad \Phi^T\Phi = I$$

Then, Σ_- and Σ_+ are transformed to:

$$\begin{aligned} \theta^{-\frac{1}{2}}\Phi^T\Sigma_-\Phi\theta^{-\frac{1}{2}} &= I \\ \theta^{-\frac{1}{2}}\Phi^T\Sigma_+\Phi\theta^{-\frac{1}{2}} &= R \end{aligned}$$

In general, R is not a diagonal matrix, but it is symmetric.

The second step is to apply the orthonormal transformation to diagonalize R:

$$Z = \Psi^T Y$$

where Ψ contains, as its columns, the orthonormal eigenvectors of R, and where Λ is a diagonal matrix whose entries are the corresponding eigenvalues of R. i.e.

$$R\Psi = \Psi\Lambda \qquad \text{and} \qquad \Psi^T\Psi = I$$

and:

$$\begin{aligned} \Psi^T\theta^{-\frac{1}{2}}\Phi^T\Sigma_-\Phi\theta^{-\frac{1}{2}}\Psi &= I \\ \Psi^T\theta^{-\frac{1}{2}}\Phi^T\Sigma_+\Phi\theta^{-\frac{1}{2}}\Psi &= \Lambda \end{aligned}$$

Thus, both matrices are diagonalized.

B.1 Optimization of the algorithm

If a numerical procedure is available which can calculate the eigensystem of a non-symmetric matrix, then the above discussion can be simplified [24, pp 34-35]. We apply the orthonormal transformation to diagonalize the matrix: $[\Sigma_-]^{-1}\Sigma_+$:

$$Y = A^T X$$

where A contains, as its columns, the orthonormal eigenvectors of $[\Sigma_-]^{-1}\Sigma+$, and where Λ is a diagonal matrix whose entries are the corresponding eigenvalues of $[\Sigma_-]^{-1}\Sigma_+$.

We then scale the columns of A such that $A^T\Sigma_- A = I$. i.e. replace each eigenvector u_i in A with:

$$\frac{u_i}{\sqrt{u_i^T\Sigma_- u_i}}$$

In our implementation, we did not have access to a routine which would calculate the eigensystem for a non-symmetric matrix, so our implementation followed the derivation in the previous section.

C Feature Selection with the SDM Transform

This section derives the "feature selection" criterion used to optimally select components of the SDM transform matrix. We wish to determine which of the vectors of A_{SDM} we can discard with minimum mean square error in the reconstruction.

The first step is to partition A_{SDM}^T such that the top m rows are the ones we plan to keep, and the bottom $k-m$ rows are the ones we plan to discard. We desire to determine the mean square error if the $k-m$ rows are discarded. We can then use this result to evaluate any such partition, and use the partition which minimizes the MSE while still discarding $k-m$ components.

We start with some notation. The symbols θ, Φ, and Ψ are as defined in Appendix B.

$$\begin{aligned}
A^T &= \Psi^T\theta^{-\frac{1}{2}}\Phi^T \\
A^{-T} &= \Phi\theta^{\frac{1}{2}}\Psi \\
A_m^T &= \text{the top } m \text{ rows of } A_{SDM}^T \\
A_k^T &= \text{the remaining } k-m \text{ rows of } A_{SDM}^T \\
A_m^{-T} &= \text{the left } m \text{ columns of } A^{-T} \\
A_k^{-T} &= \text{the remaining } k-m \text{ columns of } A^{-T} \\
Y &= A^T X \\
Y_k &= A_k^T X \\
Y_m &= A_m^T X \\
I_k &= \text{a } (k-m)\times(k-m) \text{ identity matrix}
\end{aligned}$$

If $\hat{X}$ is the reconstruction of X, the reconstruction error is:

$$\begin{aligned}
e &= X - \hat{X} \\
&= A^{-T}Y - A_m^{-T}A_m^T X \\
&= A^{-T}Y - A_m^{-T}Y_m \\
&= A_k^{-T}Y_k \\
\|e\|^2 &= < e, e >
\end{aligned}$$

$$= Y_k^T \left[A_k^{-T}\right]^T A_k^{-T} Y_k$$

Now, we concern ourselves with determining the central part of this equation:

$$\begin{aligned}
\left[A_k^{-T}\right]^T A_k^{-T} &= \left\{\left[\Phi\theta^{\frac{1}{2}}\Psi\right]\begin{bmatrix}0\\ I_k\end{bmatrix}\right\}^T \left\{\left[\Phi\theta^{\frac{1}{2}}\Psi\right]\begin{bmatrix}0\\ I_k\end{bmatrix}\right\} \\
&= \begin{bmatrix}0 & I_k\end{bmatrix}\left[\Psi^T\theta^{\frac{1}{2}}\Phi^T\right]\left[\Phi\theta^{\frac{1}{2}}\Psi\right]\begin{bmatrix}0\\ I_k\end{bmatrix} \\
&= \begin{bmatrix}0 & I_k\end{bmatrix}\left[\Psi^T\theta\Psi\right]\begin{bmatrix}0\\ I_k\end{bmatrix} \\
&= \begin{bmatrix}0 & I_k\end{bmatrix} K \begin{bmatrix}0\\ I_k\end{bmatrix} \\
&= K'
\end{aligned}$$

where K' is a $(k-m)\times(k-m)$ matrix, which is the bottom right corner of $[\Psi^T\theta\Psi]$. In particular, we note that K' is not diagonal. So,

$$\begin{aligned}
E[\| e^2 \|] &= E[Y_k^T K' Y_k] \\
&= E[X^T A_k K' A_k^T X] \\
&= E[X^T K'' X]
\end{aligned}$$

This does not simplify as in the KL case. In order to evaluate it for a given partition of A_{SDM}, we need to first compute A_{SDM}, and then compute K'' using the known quantities used when calculating A_{SDM}. Then, we need to make another pass through the training data to calculate $E[X^T K'' X]$.

Since K'' is not diagonal, the error induced by deleting one column of A_{SDM} is not independent of other columns as it was for the KL transform. Thus, all $\begin{pmatrix} k \\ m \end{pmatrix}$ permutations of A_m need to be considered, to see which rows to discard. Since each time we do this we need to make a pass through the training data, this is computationally intractable.

C.1 The Computable Method

Instead of trying all combinations of rows from A_{SDM}, we pursue the sub-optimal, but more tractable approach of minimizing the step-wise error of eliminating one row at a time. This is the so-called "greedy" approach to optimization. Therefore, we derive the error incurred by eliminating a single row from A_{SDM}.

We move the row to be eliminated to the bottom of A^T. Y_k is now a scalar, and A_k^{-T} is a column vector. The reconstruction error is:

$$\|e\|^2 = Y_k^T \left[A_k^{-T}\right]^T A_k^{-T} Y_k$$

Again, we concern ourselves with determining the central part of this equation:

$$\begin{aligned}
\left[A_k^{-1}\right]^T A_k^{-1} &= \left\{\left[\Phi\theta^{\frac{1}{2}}\Psi\right]\begin{bmatrix}0\\1\end{bmatrix}\right\}^T\left\{\left[\Phi\theta^{\frac{1}{2}}\Psi\right]\begin{bmatrix}0\\1\end{bmatrix}\right\} \\
&= \begin{bmatrix}0 & 1\end{bmatrix}\left[\Psi^T\theta^{\frac{1}{2}}\Phi^T\right]\left[\Phi\theta^{\frac{1}{2}}\Psi\right]\begin{bmatrix}0\\1\end{bmatrix} \\
&= \begin{bmatrix}0 & 1\end{bmatrix}\left[\Psi^T\theta\Psi\right]\begin{bmatrix}0\\1\end{bmatrix} \\
&= \begin{bmatrix}0 & 1\end{bmatrix}K\begin{bmatrix}0\\1\end{bmatrix} \\
&= K_{k,k}
\end{aligned}$$

i.e. we are picking the k^{th} diagonal component out of the K matrix, if we delete the k^{th} row from A_{SDM}. More generally, we pick the i^{th} diagonal component out of the K matrix if we delete the i^{th} row from A_{SDM}.

$$\begin{aligned}
E[\parallel e^2 \parallel] &= E[Y_k^T K_{k,k} Y_k] \\
&= K_{k,k} E[Y_k Y_k^T] \quad \text{Since } Y_k = Y_k^T \text{ as } Y_k \text{ is a scalar.} \\
&= K_{k,k} A_k E[X X^T] A_k^T \\
&= K_{k,k} A_k \Sigma A_k^T
\end{aligned}$$

where Σ is the covariance array calculated for the regular KL transform. More generally, for the i^{th} component:

$$\begin{aligned}
E[\parallel e^2 \parallel] &= K_{i,i} A_i \Sigma A_i^T \\
&= K_{i,i}\left[A_{SDM}\Sigma A_{SDM}^T\right]_{i,i}
\end{aligned}$$

So, we can compute all of these errors in parallel:

$$E[\parallel e_1^2 \parallel, \parallel e_2^2 \parallel, \ldots \parallel e_k^2 \parallel] = \left[\Psi^T\theta\Psi\right]_{i,i} \cdot \left[A_{SDM}\Sigma A_{SDM}^T\right]_{i,i}$$

D Three Dimensional Jump/No Jump Mesh Plots

This appendix contains the "before and after" plots of the raw 'x' data (the 'y' data is similar) in the neighborhood of all the jumps detected by the "Global Jump Search with Step Detection" routine, run when the threshold was 1.0 standard deviations. In all cases, the locations near the jumps are greatly improved, and in many instances the jump is completely eliminated. The data is displayed as three dimensional mesh plots over two dimensional contour plots.

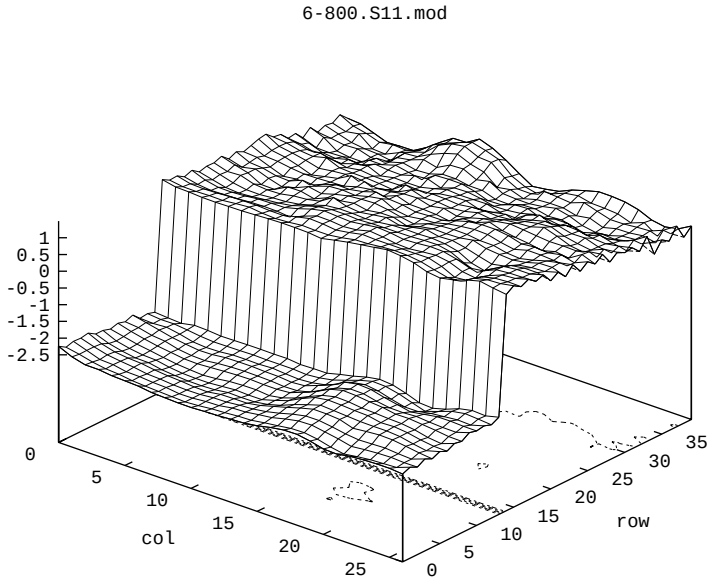


Figure 62: Jump @ Row 14 in 800 MHz, s11 Data.

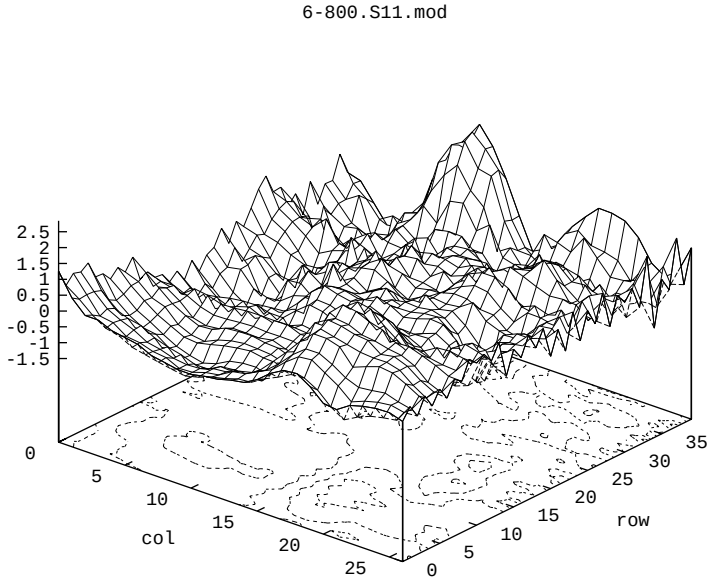


Figure 63: Data after Jump @ Row 14 in 800 MHz, s11 Data Removed.

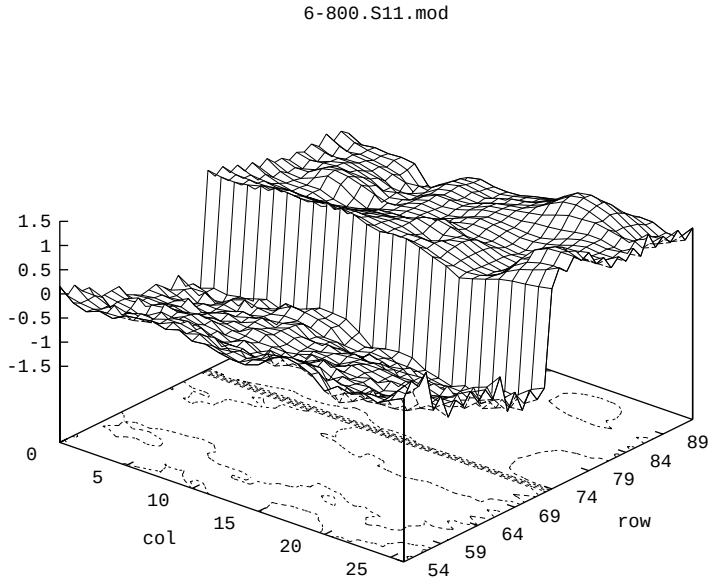


Figure 64: Jump @ Row 74 in 800 MHz, s11 Data.

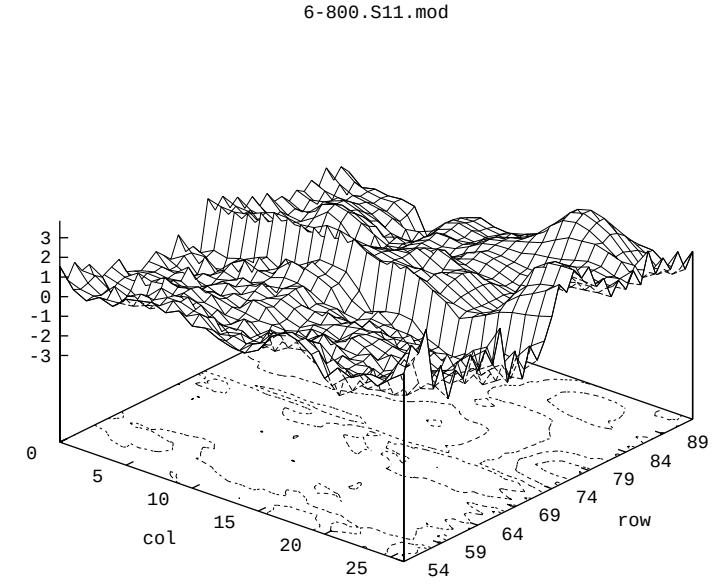


Figure 65: Data After Jump @ Row 74 in 800 MHz, s11 Data Removed.

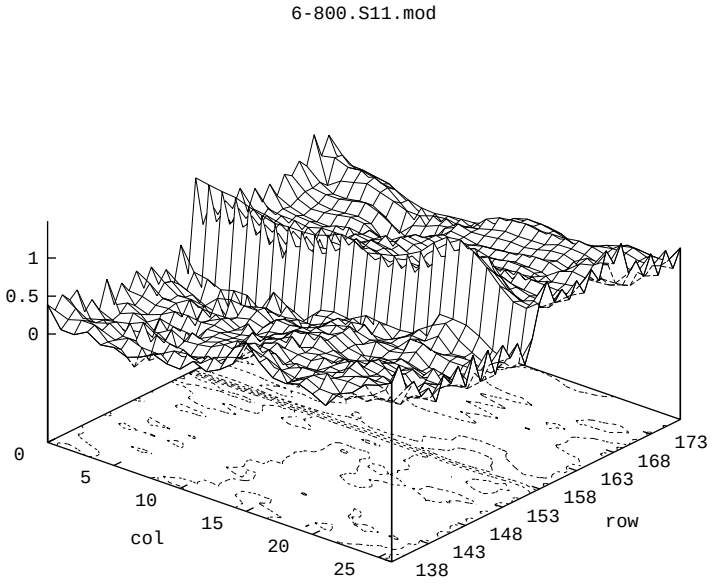


Figure 66: Jump @ Row 158 in 800 MHz, s11 Data.

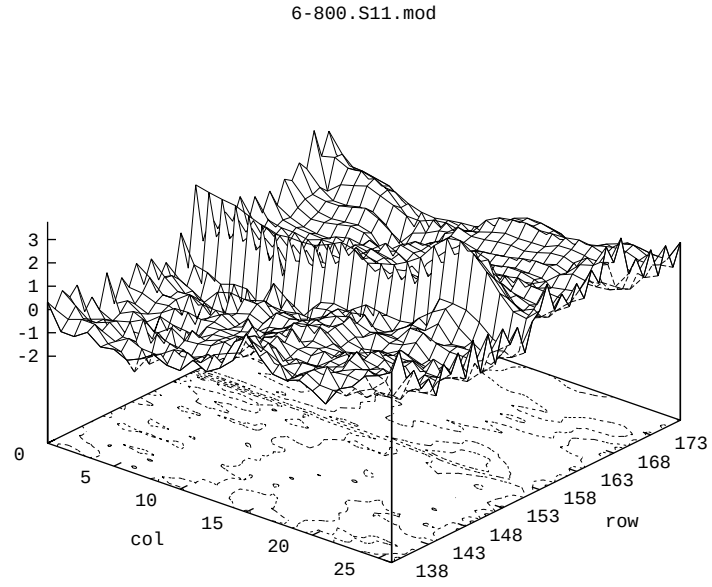


Figure 67: Data After Jump @ Row 158 in 800 MHz, s11 Data Removed.

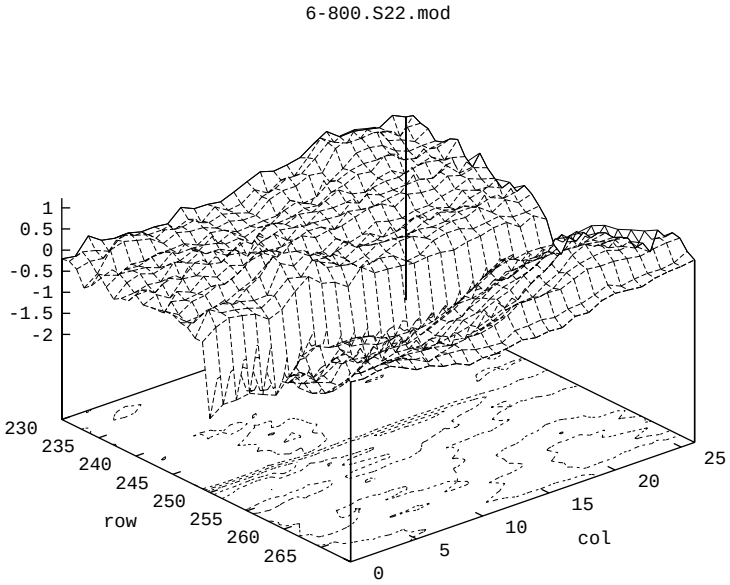


Figure 68: Jump @ Row 250 in 800 MHz, s22 Data.

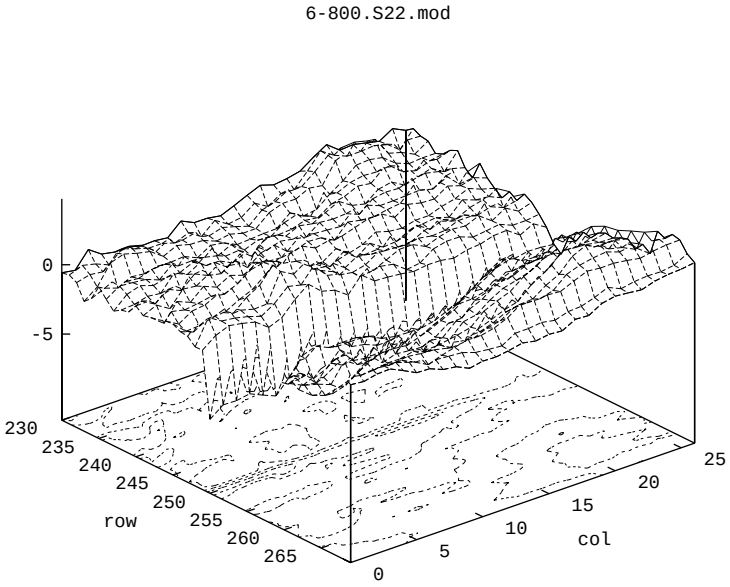


Figure 69: Jump @ Row 250 in 800 MHz, s22 Data.

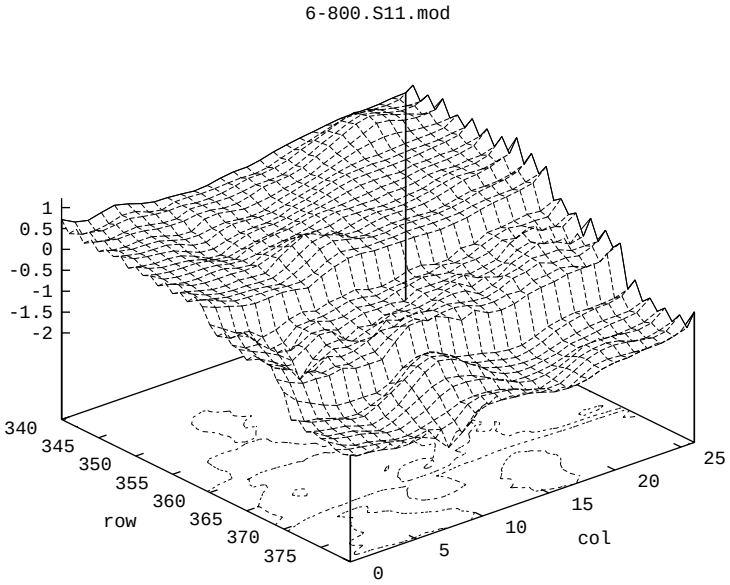


Figure 70: Jump @ Rows 360 and 370 in 800 MHz, s11 Data.

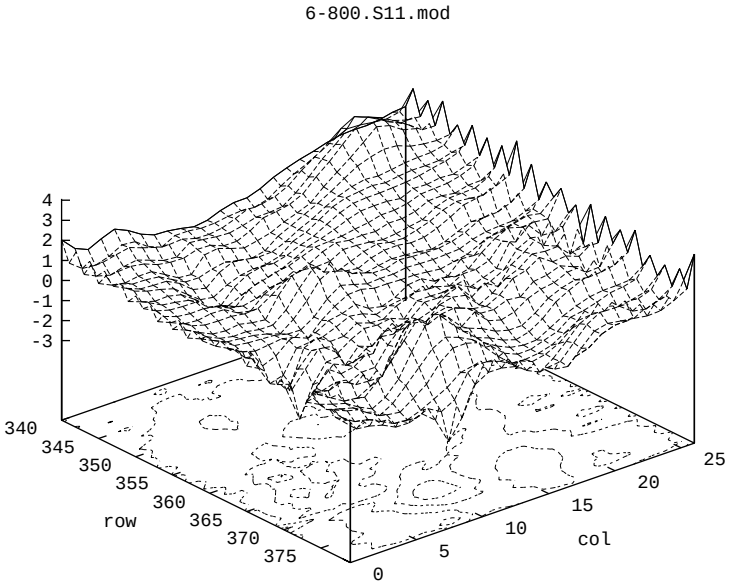


Figure 71: Data After Jumps @ Rows 360 and 370 in 800 MHz, s11 Data Removed.

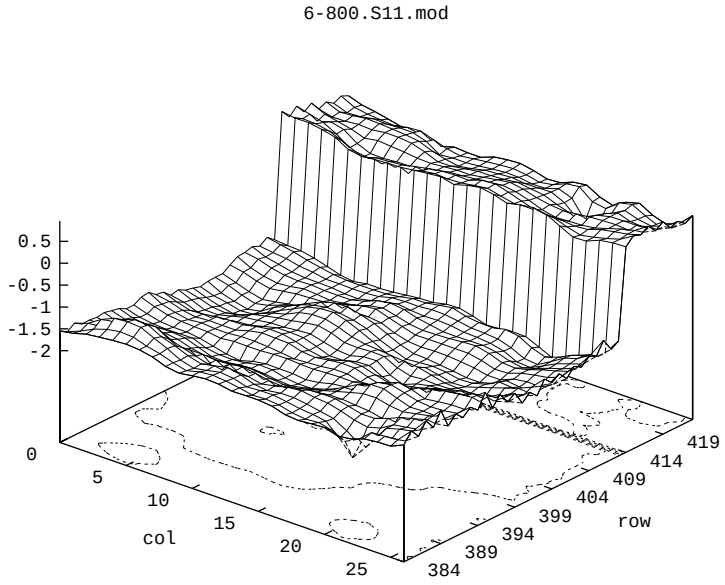


Figure 72: Jump @ Row 414 in 800 MHz, s11 Data.

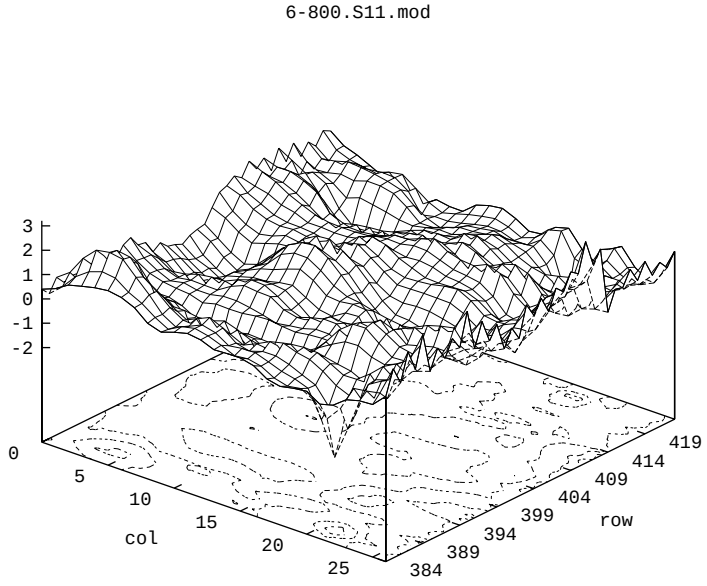


Figure 73: Data After Jump @ Row 414 in 800 MHz, s11 Data Removed.

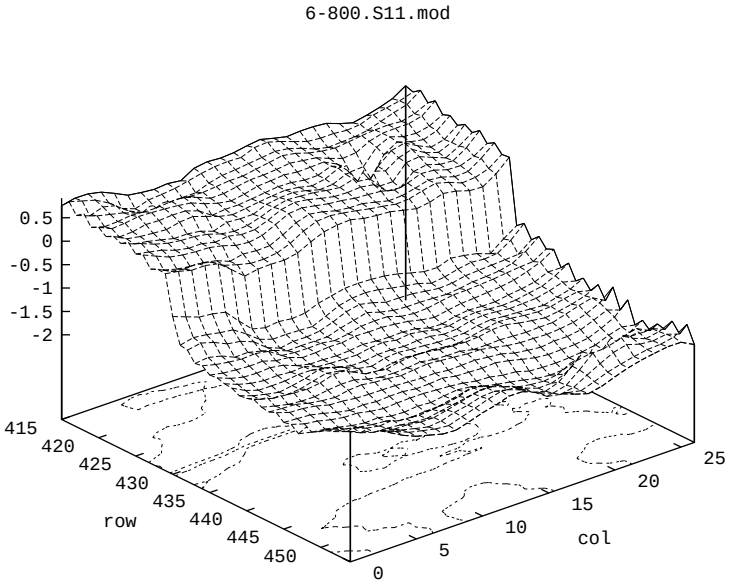


Figure 74: Jump @ Row 430 in 800 MHz, s11 Data.

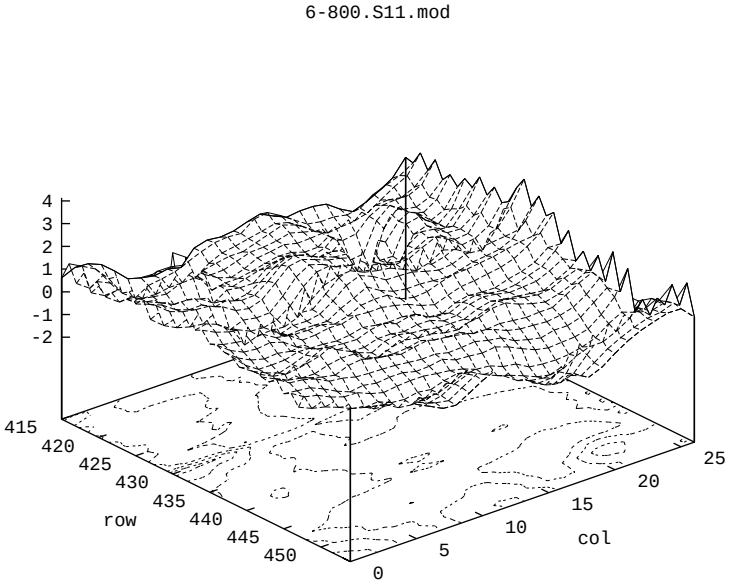


Figure 75: Data After Jump @ Row 430 in 800 MHz, s11 Data Removed.

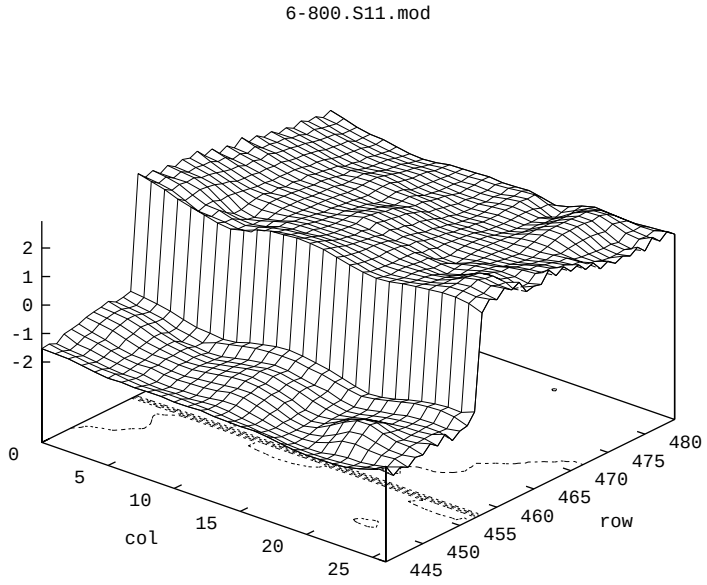


Figure 76: Jump @ Row 458 in 800 MHz, s11 Data.

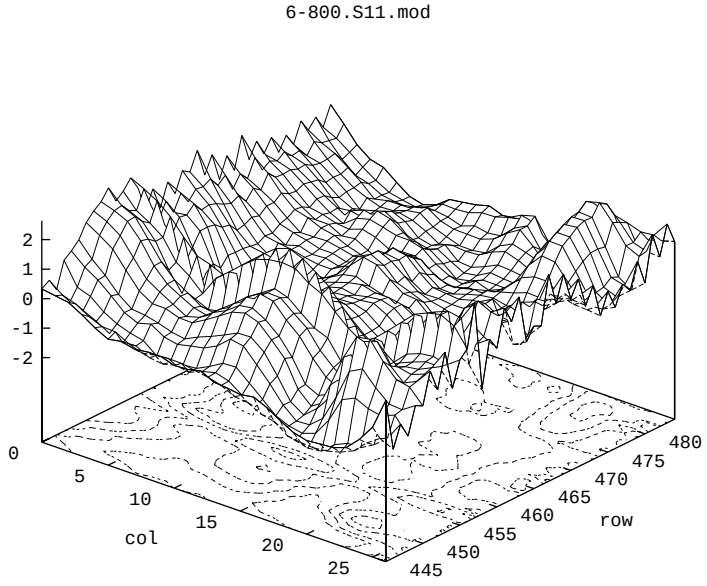


Figure 77: Data After Jump @ Row 458 in 800 MHz, s11 Data Removed.

References

[1] M. Abbasi and M. R. Sayeh, *Class of learning algorithms for multilayer perceptron*, in Proceedings of SPIE - The International Society for Optical Engineering, Applications of Optical Engineering: Proceedings of OE/Midwest '90, September 1990, pp. 237–242.

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

[3] H. I. Avi-Itzhak, *High Accuracy Correlation Based Pattern Recognition*, PhD thesis, Stanford University, Stanford, CA, May 1994.

[4] M. R. Azimi-Sadjadi, *Mine detection and classification using neural networks: New architectures and data reresentation schemes.* Contract DAAL03-86-D-0001, Colorado State University, May 1992.

[5] R. Battiti, *First- and second-order methods for learning: Between steepest descent and newton's method*, Neural Computation, 4 (1992), pp. 141–166.

[6] M. S. Bazarra, H. D. Sherali, and C. Shetty, *Nonlinear Programming: Theory and Algorithms*, John Wiley and Sons, New York, New York, 1993.

[7] D. A. Bloom, *Land mines keep wars from ever coming to an end*, The Christian Science Monitor, 87 (1995), p. 19.

[8] A. Bottoms and H. Bayless, *UN secretary general defines the land mine crisis; cites costs in human suffering; and unfavorable exchange between emplacement and removal costs*, Mine Lines; Topics in the art of mine warfare, 2 (1995), p. 18.

[9] F. R. Clague, *Summary of experiments with the separated aperture technique of dielectric anomaly detection, final report.* Report number SR-723-29-89, U.S. Army Ft. Belvoir RD&E Center, Ft. Belvoir, VA, 1989.

[10] D. DeSieno, *Adding a conscience to competitive learning*, in IEEE International Conference on Neural Networks, vol. 1, New York, 1988, (San Diego 1988), IEEE, pp. 117–124.

[11] T. Doi and G. Plett, *Contractor's progress, status, and management report for period 3/27/92-6/30/92.* Contract DAAK70-93-K-0003, Stanford University, August 1992.

[12] ——, *Contractor's progress, status, and management report for period 7/1/92-9/30/92.* Contract DAAK70-93-K-0003, Stanford University, November 1992.

[13] ——, *Adaptive neural networks for mine detection, annual report for period 4/1/92-3/31/93.* U.S. Army Contract DAAK70-93-K-0003, Stanford University, August 1993.

[14] ——, *Contractor's progress, status, and management report for period 10/1/92-12/31/92.* Contract DAAK70-93-K-0003, Stanford University, January 1993.

[15] ——, *Contractor's progress, status, and management report for period 4/1/93-6/30/93.* Contract DAAK70-93-K-0003, Stanford University, August 1993.

[16] ——, *Contractor's progress, status, and management report for period 7/1/93-9/30/93.* Contract DAAK70-93-K-0003, Stanford University, November 1993.

[17] ——, *Adaptive neural networks for mine detection, annual report for period 4/1/93-3/31/94.* U.S. Army Contract DAAK70-92-K-0003, Stanford University, May 1994.

[18] ——, *Contractor's progress, status, and management report for period 10/1/93-12/31/93.* Contract DAAK70-93-K-0003, Stanford University, January 1994.

[19] ——, *Contractor's progress, status, and management report for period 4/1/94-6/31/94.* Contract DAAK70-93-K-0003, Stanford University, August 1994.

[20] ——, *Contractor's progress, status, and management report for period 7/1/94-9/30/94.* Contract DAAK70-93-K-0003, Stanford University, November 1994.

[21] ——, *Contractor's progress, status, and management report for period 10/1/94-12/31/94.* Contract DAAK70-93-K-0003, Stanford University, February 1995.

[22] R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis*, Wiley, New York, 1973.

[23] R. Fletcher and C. M. Reeves, *Function minimization by conjugate gradients*, Computer Journal, 7 (1964), pp. 149–154.

[24] K. Fukunaga, *Introduction to Statistical Pattern Recognition*, Academic Press, New York, 1972.

[25] A. Gersho and R. Gray, *Vector Quantization and Signal Compression*, Kluwer Academic Publishers, Boston, 1992.

[26] D. M. Green and J. A. Swets, *Signal Detection Theory and Psychophysics*, Peninsula Publishing, Los Altos, California, 1988.

[27] R. A. Jacobs, *Increased rates of convergence through learning rate adaptation*, Neural Networks, 1 (1988), pp. 295–307.

[28] T. Kohonen et al., *SOM_PAK: The self-organizing map program package v 1.2.* Available via anonymous FTP to: cochlea.hut.fi (130.233.168.48), Nov 2, 1992.

[29] ——, *LVQ_PAK: The learning vector quantization program package v 2.1.* Available via anonymous FTP to: cochlea.hut.fi (130.233.168.48), Oct 9, 1992.

[30] M. A. Lehr, *Adaptive multisource decision-making: Detecting land mines with neural networks using separated aperture sensor data collected at fort belvoir.* Final Report, U.S. Army Contract DAAK70-89-K-0001, Stanford University, April 1992.

[31] D. G. Luenberger, *Introduction to Linear and Nonlinear Programming*, Addison-Wesley, Menlo Park, California, 1984.

[32] M. L. Meistrell and K. A. Spackman, *Evaluation of neural network performance by ROC analysis: Examples from the biotechnology domain*, Computer Methods and Programs in Biomedicine, 32 (1989), pp. 73–80.

[33] M. F. Moller, *A scaled conjugate gradient algorithm for fast supervised learning*, Neural Networks, 6 (1993), pp. 525–533.

[34] K. Oehler and R. Gray, *Combining image classification and image compression using vector quantization*, in Proceedings of the Data Compression Conference, Los Alamitos, CA, 1993, (Snowbird, Utah, 1993), IEEE, pp. 2–11.

[35] D. B. Parker, *Optimal algorithms for adaptive neural networks: Second order back propagation, second order direct propagation, and second order hebbian learning*, in Proceedings of the IEEE First International Conference on Neural Networks, vol. II, San Diego, CA, June 1987, pp. 593–600.

[36] K. Perlmutter, R. Gray, K. Oehler, and R. Olshen, *Bayes risk weighted tree-structured vector quantization with posterior estimation*, in Proceedings of the Data Compression Conference, Los Alamitos, CA, 1994, (Snowbird, Utah, 1994), IEEE, pp. 274–283.

[37] W. Press, B. Flannery, S. Teukolsky, and W. Vetterling, *Numerical Recipes in C*, Cambridge University Press, Cambridge, 1986, ch. 10, pp. 290–323.

[38] ——, *Numerical Recipes in C*, Cambridge University Press, Cambridge, 1986, ch. 14, pp. 523–527.

[39] L. Riggs and C. Amazeen, *Research efforts with the waveguide beyond cutoff or separated aperture dielectric anomaly detection scheme.* Internal Report #2497, U.S. Army Ft. Belvoir RD&E Center, Ft. Belvoir, VA, 1989.

[40] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Learning internal representations by error propagation*, in Parallel Distributed Processing, D. E. Rumelhart and J. L. McClelland, eds., vol. 1, The MIT Press, Cambridge, MA, 1986, ch. 8.

[41] D. E. Rumelhart and J. L. McClelland, eds., *Parallel Distributed Processing*, vol. 1 and 2, The MIT Press, Cambridge, MA, 1986.

[42] J. Sammon, Jr., *A nonlinear mapping for data structure analysis*, IEEE Transactions on Computers, C-18 (1969), pp. 401–409.

[43] R. Schalkoff, *Pattern Recognition: Statistical, Structural and Neural Approaches*, Wiley, New York, 1992.

[44] K. A. Spackman, *Signal detection theory: Valuable tools for evaluating inductive learning results*, in Proceedings of the Sixth International Workshop on Machine Learning, 1989, pp. 160–163.

[45] D. F. Spect, *Probabalistic neural networks*, Neural Networks, 3 (1990), pp. 109–118.

[46] D. Torrieri, *Personal communication*, 1995.

[47] N. Ueda and R. Nakano, *A competitive & selective learning method for designing optimal vector quantizers*, in Proceedings of ICNN - IEEE International Conference on Neural Networks, March 1993, pp. 1444–1450.

[48] P. D. Wasserman, *Neural Computing: Theory and Practice*, Van Nostrand Reinhold, New York, 1989.

[49] ——, *Advanced Methods in Neural Computing*, Van Nostrand Reinhold, New York, 1993.

[50] P. Werbos, *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*, PhD thesis, Harvard University, Cambridge, MA, August 1974.

[51] R. Wesel and R. Gray, *Bayes risk weighted vq and learning vq*, in Proceedings of the Data Compression Conference, Los Alamitos, CA, 1994, (Snowbird, Utah, 1994), IEEE, pp. 400–409.