Calibration of Online Analyzers Using Neural Networks

FINAL REPORT

to the

United States Department of Energy

By

Rajive Ganguli, Ph.D., P.E., Daniel E. Walsh, M.S., P.E., Shaohai Yu, M.S.

**Mineral Industry Research Laboratory
School of Mineral Engineering
University of Alaska Fairbanks
Fairbanks, Alaska 99775**

December 5, 2003

Contract #: DE-FC26-01NT41058

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

ABSTRACT

Neural networks were used to calibrate an online ash analyzer at the Usibelli Coal Mine, Healy, Alaska, by relating the Americium and Cesium counts to the ash content. A total of 104 samples were collected from the mine, with 47 being from screened coal, and the rest being from unscreened coal. Each sample corresponded to 20 seconds of coal on the running conveyor belt.

Neural network modeling used the quick stop training procedure. Therefore, the samples were split into training, calibration and prediction subsets. Special techniques, using genetic algorithms, were developed to representatively split the sample into the three subsets.

Two separate approaches were tried. In one approach, the screened and unscreened coal was modeled separately. In another, a single model was developed for the entire dataset. No advantage was seen from modeling the two subsets separately.

The neural network method performed very well on average but not individually, i.e. though each prediction was unreliable, the average of a few predictions was close to the true average. Thus, the method demonstrated that the analyzers were accurate at 2-3 minutes intervals (average of 6-9 samples), but not at 20 seconds (each prediction).

Table of Contents

EXECUTIVE SUMMARY	1
I. INTRODUCTION	3
II. SAMPLING	5
III. NEURAL NETWORKS MODELING STRATEGIES AND BACKGROUND .	13
IV. NEURAL NETWORK MODELING	33
V. PROGRAMMING	45
VI: CONCLUSIONS AND RECOMMENDATIONS	48
ACKNOWLEDGMENTS	50
REFERENCES	51
Appendix – I: Data and Models for Kohonen-based-Data-Division Models	53
Appendix – II: Data for GA-based-Data-Division Models	68
Appendix-III: Program Code.....	78

EXECUTIVE SUMMARY

A neural network was applied for the calibration of an online analyzer at the Usibelli Coal Mine, Healy, Alaska, i.e. neural networks were used to relate the Americium-241(Am) and Cesium-137 (Cs) counts to the ash content. 104 samples were collected as part of the project. The samples were collected from coals from different parts of the mine over a period of several months. Out of the 104 samples, 47 were taken from screened coal, while the rest were from unscreened coal. The “true” ash content of the samples was determined in a laboratory.

Each sample corresponded to 20 seconds of coal on a conveyor belt moving at full speed (roughly 3 tons). Since that was too much to process in a laboratory, each sample was sub-sampled at five locations to estimate the true ash value. For 20 of the 104 samples, the individual sub-samples were analyzed in the laboratory rather than being pooled, to obtain an estimate of the variance within a sample. It was observed that the in-sample variance was proportional to the mean ash content of the sample. Samples below 12% ash content (low ash) had a variance of 0.67, while those above 12% ash content (high ash) had an average variance of 1.72. Therefore, the 95% confidence interval for the mean ash content was ± 1.00 for low ash samples and ± 1.61 for high ash samples. This highlighted one of the major issues in this project: noise in the data.

The “quick stop” training procedure was followed for neural network modeling. Therefore, the data had to be split into three subsets, training, calibration and prediction. Using procedures involving genetic algorithms and Kohonen networks, the data was split so that the three subsets were similar to each other. The resultant subsets were statistically close to each other in all categories,

Am, Cs and ash. While testing the efficacy of neural networks, extended modeling was also done to examine if there was an advantage to modeling the screened and unscreened coal separately.

Neural network predictions were accurate at 2-3 minute averages though individual predictions were not reliable (low R^2 of predictions and insignificant bias). Modeling the screened and unscreened data separately did not result in any additional benefit.

The project also required significant amount of programming. C++ programs were written to process the analyzer readings, and then later, to apply neural networks.

I. INTRODUCTION

The United States Department of Energy awarded the grant (DE-FC26-01NT41058) to study the application of neural networks in calibrating online analyzers to improve their measurements. Current industry practice involves the use of regression type techniques to relate the Americium (Am) and Cesium (Cs) counts to the quality parameter being measured. The project studied the use of neural networks (instead of regression) in modeling the relationship between the analyzer scintillation counts (Am and Cs) and the quality parameter (ash).

The study was carried out at the Usibelli Coal Mine (UCM), Healy, AK. The mine employs a Berthold dual energy type ash analyzer. The general principles of the analyzer are described briefly below.

1.1. The On-line Analyzer

The main components of the analyzer at the mine consist of the two gamma ray sources (Am and Cs) located below the belt with their corresponding scintillation counters located above the belt (Figure 1.1). The sources bombard the coal stream with gamma radiation. The scintillation counters measure the amount of gamma radiation which is not absorbed by the air gap, belt and coal.

Figure 1.1. A DUET On-line Analyzer (after Kamada et al., 1986).

The amount of radiation that passes through the coal stream depends on the elemental composition of coal, with heavier elements absorbing more radiation than the lighter elements. Therefore, the scintillation counts can be used to estimate the elemental composition of coal, including ash content. It is obvious, however, that other factors can complicate the relationship between the elemental composition of coal and the scintillation counts. Ganguli (2001) cites coal particle size distribution, moisture content, and chemical composition as some of the complicating factors. It is because of these complicating factors that a simple linear regression often fails to properly relate the Am and Cs counts to ash content in run-of-mine coal.

II. SAMPLING

The sampling exercise was conducted at the Usibelli Coal Mine (UCM), Healy, Alaska. The data collected consisted of online analyzer (OLA) readings (Americium (Am) and Cesium (Cs) scintillation counts) with their corresponding laboratory ash value.

2.1. The Sampling Process

The schematic of the coal handling system at UCM relevant to the sampling exercise is given in Figure 1. The coal from the mine is loaded onto the main belt at Tipple A, while it is unloaded for rail transport at Tipple B. The gradient of the conveyor belt is quite steep except for the inside of the Tipple B, where it is flat. Since the belt is in a windy region, most of the conveyor belt is enclosed to prevent spreading coal dust. The OLA is mounted close to Tipple A as shown. The Am and Cs counts are displayed in the control room on a computer screen.

Figure 2.1. A simple schematic of the coal flow at UCM (after Yu et al, 2003)

Under operational conditions, the OLA scans the coal at full belt speed (3 m/s). Therefore, to match the operating conditions, it was decided that the sampling would be done at full belt speed as well. Given the lack of an automatic sampler at the mine, this made the sampling difficult, as isolating the block that was scanned by the OLA at full belt speed was not easy. The block had to be isolated so physical samples could be obtained to determine the laboratory ash value of the sample. This was implemented as described below.

One person (person A) stood by the OLA in radio contact with the control room attendant (person B) who was monitoring the real time Am and Cs counts. When both were ready, person A marked the beginning of the sample section by throwing a fistful of lime on the coal in the moving conveyor belt. He simultaneously radioed person B so that the OLA readings could be saved. After 20 seconds from the start, person A threw another fistful of lime onto the coal stream to mark the end of sample and simultaneously radioed person B in order to mark the end of Am and Cs readings. Thus, the block of coal between the two lime marks constituted the coal scanned by the OLA during the sampling period, while the OLA readings saved during the 20 seconds constituted the corresponding Am and Cs readings. The belt was stopped at the appropriate location so that physical samples could be obtained from the sample block of coal. This is discussed in the next section.

The length of each sample (20 seconds or 60 m) was determined by the configuration of Tipple B (see Figure 2.1). A longer sample, though preferred, would have resulted in a significant portion of the sample being outside of Tipple B. Repeated sampling from the steep portion of the covered conveyor belt proved difficult.

The above was done 104 times for the 104 samples. There were 40 Am and Cs readings for each sample as the OLA outputted one reading every 0.5 seconds. A C++ program was written to

extract the relevant Am and Cs readings from the text files containing the analyzer readings. Chapter V discusses the programming aspects for the project in more detail.

Out of the 104 samples, 57 were collected from unscreened coal, while the rest were collected from screened coal. Coal screening is optional at the mine and involves removal of $-3/8$ inch particles. The physical sampling is discussed next.

2.1.1. Obtaining Laboratory Values

The true ash content of the sample was determined in the laboratory. However, given that the sample was large (60 m of fully loaded belt), and due to the scope of the study, it was not possible to process the entire sample length in the laboratory. Therefore, it was decided to sub-sample the 60 m (~200 feet) sample in five locations to obtain the average ash value of the sample. Each sub-sample was obtained by taking a 25.4 cm (10 inch) slice of coal from the belt. The sub-samples were located 10 m apart, the first being 10 m from the beginning of the sample (Figure 2.2). Given the distance from the lime markers, the sub-samples were not contaminated by lime.

Figure 2.2. A typical sample collection (after Yu, 2003)

The five sub-samples were processed using two different methods to obtain the average ash of the 60 m sample. In the first method, the five sub-samples were physically combined, crushed to – 0.635 cm (-1/4 inches), split (to obtain a representative quantity for the lab) and analyzed in the UCM laboratory to obtain the ash value. In the second method, each sub-sample was separately crushed, split and analyzed to obtain individual ash values, which were averaged to obtain the ash value for the entire sample. The second procedure, therefore, gave an estimate of the variance within the sample. Twenty samples were prepared using the second procedure. Figure 2.3 gives the mean-standard deviation plot for the twenty samples.

Figure 2.3. Mean-standard deviation relationship for the samples

2.2. Confidence Interval of Each Sample

When a group of samples are used to estimate the mean (such as here, where the true ash of a sample is estimated from five sub-samples), sampling theory principles can be used to estimate a confidence interval for the mean. In other words, the confidence around the estimated ash values can be determined using information obtained from the twenty samples whose ash content was obtained by averaging the ash content of the five sub-samples.

Per sampling theory, the variance of the mean estimate is given by:

$$\hat{V}(\bar{y}_{sy}) = \frac{s^2}{n} \left(\frac{N-n}{N} \right) \quad (2-1)$$

In which,

$$s^2 = \frac{\sum_{i=1}^n (y_i - \bar{y})^2}{n-1} \quad (2-2)$$

Where:

N = population size

n = the number of samples

s^2 = the sample variance

y_i = the individual sample value

$\bar{y}$ = the sample mean

In our case, each of the 5 sub-samples had a width of 25.4 cms out of a total width of 60 m.

Therefore, $n=5$ and $N = 60/0.254$ or 240.

The above principles could be used to estimate the confidence interval of the laboratory ash values. However, in this case, one factor complicated the issue: the relationship between the magnitude of the measured value and its standard deviation. From Figure 2.3, it is apparent that the mean and the standard deviation are related. This implies that the higher ash values would have a larger error range around the mean estimate. Therefore, to obtain a better estimate of the error range around the mean for the various ash values, the 104 samples were split into two categories, high ash (>12%) and low ash (<=12%). The error ranges were then determined separately for the two groups using the twenty samples whose in-sample variability was known (Section 2.1.1). This was done by splitting the twenty samples into two groups, low ash (<12%) and high ash (>12%). The average variance for each group was used to obtain the confidence interval for that group.

The variance of the low ash group was 0.67, while that of the high ash group was 1.72. Therefore, the variance for the mean, for the low ash group, was given by:

$$\hat{V}(\bar{y}_{sy}) = \frac{.67}{5} \left(\frac{240 - 5}{240} \right) = 0.13$$

Therefore, the estimate for a sample had a 95% confidence interval of approximately $\pm 2.78 \cdot \sqrt{0.13}$ or ± 1.00 , where the 2.78 refers to the “Student t” value for 4 degrees of freedom. Similarly, the high ash samples had a mean whose 95% confidence interval was ± 1.61 .

2.3. Analysis of Variance (ANOVA)

The variance within the samples for the twenty samples were tested to determine if the samples came from the same population. Given that the samples were obtained from different seams and over a wide range of ash content (to ensure that the neural network was exposed to the entire range), one would expect for the null hypothesis to fail in ANOVA. In an ANOVA test, the null hypothesis is that the sample means are identical. The alternate hypothesis is that there is at least one mean that is different.

$$H_0: \mu_1 = \mu_2 = \dots = \mu_{20}$$

$$H_a: \text{not all } \mu_i \text{ are equal}$$

The test statistic for the test, F^* , is the ratio of the variance between treatments (groups) and variance within treatments. If $F^* \leq F(1-\alpha; r-1, n_T-r)$, it can be concluded H_0 holds. Else, H_a holds.

In the above, r is the number of groups under study (or 20), n_T is the total number of data under study (or 100).

Assuming $\alpha = 0.05$, $F(1-\alpha; r-1, n_T-r) = F(0.95; 19, 80) = 1.73$

The ANOVA table is given below:

Dependent Variable: ash

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Between groups	19	578.9120840	30.4690571	25.48	<.0001
Within groups	80	95.6772800	1.1959660		
Corrected Total	99	674.5893640			

The F ratio of 25.48 indicates the failure of the null hypothesis since it far exceeds 1.73.

III. NEURAL NETWORKS MODELING STRATEGIES AND BACKGROUND

3. 1. Approach

The calibration exercise was designed to relate the Am and Cs counts to the ash content using a neural network. Two different approaches were tried

- i. Model the screened and unscreened coal data separately. The mine operator would have to choose the most appropriate model when using this approach.
- ii. Model the combined data set. In this approach, the mine operator would not have to choose the appropriate coal type.

It is natural to expect that two separate models, custom built for two different particle sizes, will outperform a single model that is applied to both particle size distributions. However, it is also possible that there may be no distinct advantage to using a separate model for separate particle size distributions. Therefore, both of these approaches were tried.

3.2. Modeling Concepts Applicable to All Neural Network Modeling Exercises

The applicability of a developed numerical model (including neural networks) to data beyond the initial sample set is always a concern. Applicability beyond the initial sample set requires a) the sample set to be representative of the process being sampled and b) the developed model to be representative of the sample set. Typically, every attempt is made to ensure that the first condition is met. Therefore, this is not discussed any further in this report. However, given the cost of sampling

and time constraints, often times the sample set is barely representative. This has implications when trying to meet the second condition.

Meeting the second condition may require two steps (any or all) to be taken during the model development phase:

- i. When the sample set is split into modeling and evaluation subsets, ensure that the two subsets are “similar.” This can be a challenge in datasets that are not abundant. In sparse datasets, random subdivision of the dataset may not result in the subsets being similar. If the two subsets are not similar, then the model is developed in one type of data and evaluated on another. Obviously, any evaluation in such a condition is misleading. When the two subsets are similar, then each would also be similar to the properties of the entire dataset. Two ways of ensuring similarity of subsets is discussed later.
- ii. Even when the subsets are similar, one must be careful in developing the model. Typically, many models can be fit to a given dataset, with varying types of performances. While one model may outperform all others in one metric of success, some other model may outperform it in another metric of success. This happens because while one model “understands” one aspect of the data very well (allowing it to outperform all other models in the resulting success metric), another model may understand another aspect of the data. Combining the models appropriately (as discussed later) may allow one to exploit the strengths of the different models.

3.3. Division of Data

The data was subdivided into three parts, training, calibration and prediction. The training and calibration subsets constituted the modeling subset, while the prediction subset was the evaluation subset. The reason for requiring three subsets is discussed in the quick stop training subsection later (Section 3.5.5). Note that “data” here implies screened, unscreened and combined, as discussed in Section 3.1. As noted above, it was required that the subdivision result in subsets that are similar.

3.3.1. Definition of Similar

Two definitions of similarity were applied in this project:

- i. Two subsets are deemed similar if their mean, variance, low and high were close. The popular *t-test* (Hines et al, 1980) could be used to verify if two subsets were indeed close.
- ii. The second definition is more involved as laid out by Bowden et al (2002). The sample space is first divided into statistically distinct groups. This can be done various ways such as using Kohonen networks or genetic algorithms. The members of the three subsets are then randomly selected from each of the distinct groups. The extent of contribution of each group to a subset depends on the relative size of the subset. For example, if the training subset is to be half the sample set, then each group contributes half of its members to the training subset.

- Sparseness of the data plays a role here as well. In sparse datasets, the group sizes will be small. One has to balance the needs of the subsets with the size of the groups. For example, training subsets should contain the largest range in data. Therefore, the extrema in each group is sent to the training subset. Also, since the calibration subset is used to indicate general performance (see subsection on quick stop training), its composition is carefully controlled so that it is representative of the process. Thus, careful selection (of subset members) is substituted for random selection in sparse data.

3.3.2. Obtaining Similar Subsets Using Genetic Algorithms (GA)

In this project, GA was used to split the sample set into three statistically similar subsets. GA's are (Goldberg, 1999) "search algorithms based on the mechanics of natural selection and natural genetics." The basic GA works by randomly generating a group of possible solutions (population) that are, over time (i.e. over generations), improved by a process similar to evolution, to obtain the desired solution. This is done in four steps (the reader is referred to Goldberg (1999) for a more thorough presentation of GA.):

- i. Generate a random population of solutions to start the process (i.e. create the initial generation).
- ii. Evaluate the solutions using some criteria to obtain their relative merit. In this case, the criteria used are the closeness of the statistical properties of the three subsets (mean, variance, low and high). A solution is rated higher over another if its three subsets are closer than for the other solution.
- iii. Randomly select solutions from the old population, such that the better solutions have a higher chance of getting selected, to begin moving towards a new generation. Select with replacement, i.e. a particular solution may be selected multiple times. Therefore, bad solutions get discarded while good solutions have multiple copies in the new generation.
- iv. Select pairs (“parents”) of solutions from the generation and probabilistically (i.e. does not happen all the time) conduct crossover and mutation. Crossover implies combining (“mating”) the characteristics of the two parents to obtain a pair of “children” solutions. Each child solution has traits from both parents. In the second step of the process, mutate each child, i.e. randomly alter the child (“random mutation of a species”). The new generation is now complete. Go to step ii.

The exact implementation for unscreened coal is given below. The implementation for screened coal and the combined group is identical, except for the actual data.

- 1) Obtain the first generation by creating the initial population of 50 solutions. This is done by randomly shuffling the 58 samples 50 times. In any solution (for unscreened coal), the first 42 are considered part of training subset, the next 8 part of the calibration subset, while the last 8 is considered part of the prediction/evaluation subset.
- 2) Evaluate each solution of the population based on the closeness or similarity between the subsets. Similarity between two subsets was obtained by measuring the closeness of their means and variances (low and high were not incorporated as the sample size was very small). As this determination (shown below) is similar to the popular t-statistic computation, this criteria was called the ‘modified t-statistic’. For a given solution, the modified t-statistic was the sum of the modified t-statistics computed for the nine combinations of the three groups (training, calibration and prediction): Am_Training and Am_Calibration, Am_Training and Am_Prediction, Am_Calibration and Am_Prediction, Cs_Training and Cs_Calibration, Cs_Training and Cs_Prediction, Cs_Calibration and Cs_Prediction, Ash_Training and Ash_Calibration, Ash_Training and Ash_Prediction, Ash_Calibration and Ash_Prediction.

$$\text{Modified tstat} = \frac{\text{abs}(\text{mean1} - \text{mean2})}{\text{sqr}t(\frac{\text{var 1}}{n1} + \frac{\text{var 2}}{n2})} + \frac{\text{abs}(\text{var 1} - \text{var 2})}{(\frac{\text{var 1} + \text{var 2}}{2})} \quad (3.1)$$

where mean1, var1 and n1 are the mean, variance and number of elements of the first group, while mean2, var2 and n2 are the mean, variance and number of elements of the second group.

The best solution for a given generation (the solution with the lowest modified tstat) is preserved.

- 3) Randomly select 50 solutions from the population such that the better solutions (the ones with the lowest modified tstats) have a higher probability of selection. A solution maybe selected multiple times. This step gets rid of the poor solutions.
- 4) The next step consists of crossover and mutation. Since there was no easy way to implement crossover, crossover was not implemented. As will be seen later, this had no effect on the final solutions. Mutation was conducted with a probability of 0.1, i.e. each solution had a 10% chance of being mutated. Mutation was implemented by swapping two arbitrarily selected samples (say sample #37 with sample #51). Therefore, in some cases of mutation, this resulted in shifting of one sample from one subset to another.

After considering each solution for mutation, evolution of the current generation is complete.

- 5) Go to 2. Continue till 5000 generations are completed. The final solution is the best out of the last generation.

Tables 3.1, 3.2 and 3.3 give the statistics of the three subsets for each of three cases (screened, unscreened and combined).

Table 3.1. Statistics of the three subsets obtained from GA for screened data

	Sample Points	Ash, % Mean	Ash, % Std. Dev.	Am, % Mean	Am, % Std. Dev	Cs, % Mean	Cs, % Std. Dev
Training Data Set	32	12.21	2.43	14965	1929	23471	8757
Calibration Data Set	7	12.35	2.52	14892	1928	23182	7403
Prediction Data Set	7	12.25	2.52	14943	1921	23271	8921

Table 3.2. Statistics of the three subsets obtained from GA for unscreened data

	Sample Points	Ash, % Mean	Ash, % Std. Dev.	Am, % Mean	Am, % Std. Dev	Cs, % Mean	Cs, % Std. Dev
Training Data Set	41	13.14	2.88	10371	2812	10871	7396
Calibration Data Set	8	13.12	2.92	10381	2741	10599	7906
Prediction Data Set	8	13.14	2.87	10348	2827	10590	7283

Table 3.3. Statistics of the three subsets obtained from GA for combined data

	Sample Points	Ash, % Mean	Ash, % Std. Dev.	Am, % Mean	Am, % Std. Dev	Cs, % Mean	Cs, % Std. Dev
Training Data Set	60	12.72	2.42	12365	3365	16302	10109
Calibration Data Set	20	12.73	2.43	12416	3368	16296	10189
Prediction Data Set	24	12.72	3.47	12378	3300	16393	10069

Note that the training subset for the combined data contained 60 samples and not 73 (41+32).

Similarly, the calibration set did not contain 15 samples (7+8). This is because the GA could not come up with subsets that were statistically close at those sizes.

3.3.3. Obtaining Similar Subsets From Distinct Groups

In this approach, the sample set is first divided into various distinct groups. Members of the training, calibration and prediction subsets are then selected from each of the groups. Two groups are considered distinct when none of their corresponding categories (Am, Cs and Ash) are statistically similar. A data set may be divided into various distinct groups using a variety of techniques such as

genetic algorithms and Kohonen networks (used here). In genetic algorithms, the algorithm earlier can be applied except with two modifications: i) the number of groups will be different from three and ii) the best solution is the one with the highest modified tstat (the solution where the subgroups are the farthest from each other). The reader is referred to Kohonen (1984) for a description of how Kohonen networks work.

The screened and unscreened datasets were split into various groups (using Kohonen networks), as there was no way to determine theoretically how many number of groups the data should be divided into. Each dataset was divided into 3 – 10 groups. However, in some cases, the resultant groups were not statistically significant. These cases were discarded. The training, calibration and prediction subsets were then built using members obtained from the various groups. The combined data was, however, handled differently. Its training, calibration and prediction subsets were obtained by simply merging the corresponding training, calibration and prediction subsets of the screened and unscreened data. This was done so that one could compare the joint screened/unscreened model performances with that of the single model fitted to the combined data.

For screened dataset, the best model occurred for four groups. Table 3.4 gives the statistics of the four groups. For unscreened data, the best model occurred for three groups (Table 3.5). The best models are discussed at a later section.

Table 3.4: Characteristics of the four subgroups for screened data

Group	Sample Points	Ash, % Mean	Ash, % Std. Dev.	Am, % Mean	Am, % Std. Dev.	Cs, % Mean	Cs, % Std. Dev.
Group 1	5	11.7	1.7	19214	1466	43429	7579
Group 2	15	11.6	1.7	15562	427	25332	2260
Group 3	16	13.1	2.9	14326	346	20237	1263
Group 4	11	12.0	2.4	12504	1401	14664	3177

Table 3.5: Characteristics of the three subgroups for unscreened data

Group	Sample Points	Ash, % Mean	Ash, % Std. Dev.	Am, % Mean	Am, % Std. Dev.	Cs, % Mean	Cs, % Std. Dev.
Group 1	23	12.9	3.0	8005	1734	5772	1978
Group 2	26	13.4	2.6	10932	753	10889	1856
Group 3	9	12.7	3.2	14581	2533	22815	11000

The selection of training, calibration and prediction subsets from the various subgroups was not straightforward. It had to meet the following constraints:

- Each group needed to contribute 70% of its samples to the training subset and 15% each to calibration and prediction subsets. Small group sizes may lead to slightly different contributions than one cited above.
- The training subset should see the entire range of data. Therefore, the extrema should be sent to the training subset.
- In each group, the values at the mean, the values at one standard deviation away from the mean, and extreme values go to the training data set. (see Figure 3.1)
- Thereafter, other values at the mean are put in the calibration data set.
- Remaining values are allocated to prediction data set.

Figure 3.1. Data allocation in each group

The statistics of the training, calibration and prediction subsets for the best models is given in Tables 3.7 and 3.8

Table 3.7. Statistics of the three subsets obtained from 4 groups for screened data

	Sample Points	Ash, % Mean	Ash, % Std. Dev.	Am, % Mean	Am, % Std. Dev	Cs, % Mean	Cs, % Std. Dev
Training Data Set	33	12.14	2.48	14687	1969	22502	7795
Calibration Data Set	7	12.20	2.22	15012	2513	23946	11227
Prediction Data Set	7	12.53	2.43	15211	2463	24576	11170

Table 3.8. Statistics of the three subsets obtained from 3 groups for unscreened data

	Sample Points	Ash, % Mean	Ash, % Std. Dev.	Am, % Mean	Am, % Std. Dev	Cs, % Mean	Cs, % Std. Dev
Training Data Set	42	13.22	2.95	10328	2911	10892	7698
Calibration Data Set	8	13.20	2.45	10672	2708	11337	7978
Prediction Data Set	8	12.36	2.70	10053	1990	9131	4163

Table 3.9. Statistics of the three subsets for combined data

	Sample Points	Ash, % Mean	Ash, % Std. Dev.	Am, % Mean	Am, % Std. Dev	Cs, % Mean	Cs, % Std. Dev
Training Data Set	75	12.67	2.5	12214	3326	15892	9699
Calibration Data Set	15	12.65	3.65	12678	3464	17173	11026
Prediction Data Set	15	12.93	2.53	12667	3357	17004	11272

Since the combined data was obtained by pooling the screened and unscreened subsets, the training, calibration and validation subsets are not very close in all categories (see ash standard deviation).

3.4. Summary of the various modeling procedures

From above, it is apparent that modeling had to be (and was) done on six different (versions of the) datasets. Three of the datasets used genetic algorithms for splitting the data while the other three used Kohonen networks. Out of the six versions, there were two each for unscreened data and screened data, and two for combined. At the conclusion of the modeling on the six datasets, the pooled performance of the screened and unscreened models was compared with the performance of the combined data model to see if there was any advantage to modeling the screened and unscreened data separately.

3.5. Background on Neural Networks

Artificial Neural Networks (ANN) are structures based on the architecture of the brain. As in the brain, information processing is done in parallel using a network of neurons. Therefore, ANNs have capabilities that go beyond algorithmic programming. They are very good for pattern association and pattern recognition tasks. It is, therefore, no surprise that they are widely used for real-time control and predictive tasks.

3.5.1. Architecture

Neural networks process information in interconnected layers. A layer is simply a group of elements. There is the input layer consisting of the inputs, the one or more *hidden layers* consisting of a suitable number of elements, and the output layer consisting of the outputs. Thus, a neural network has three or more layers (typically three or four). The network shown in Figure 1 has three layers. The various terms in the figure will be explained eventually in this section.

Figure 3.2. A neural network with four inputs, one hidden layer and three outputs (after Ripley, 1996)

The inputs (elements in the input layer) are first manipulated (explained later) to determine the value of the elements of the hidden layer. It will be pointed out that while the input layer and the output layer have a fixed number of elements for a given problem, the number of elements in the hidden layer is arbitrary. For the problem at hand, there will be two elements in the input layer (or two inputs), Am and Cs readings, while the output layer will have only one output, the percentage ash content. This is implying that the lone output will be determined from the two inputs.

The software (Neuroshell2[®]) used in this project, displays the architecture using a slightly different terminology, called slabs. A slab is simply a group of neurons. For example, Figure 3.2 would be displayed as shown in Figure 3.3. The thick arrows show the connectivity between neurons. A connection arrow implies that all neurons in the origin slab are connected to all neurons in the destination slab. The number inside the hidden layer slab shows the number of neurons in that slab.

Figure 3.3. Architecture description using slabs

3.5.2. Mechanics of the neural network

Each input x_i is assigned a weight w_{ij} and connected to element j of the hidden layer. Each input is connected to each node of the hidden layer just as each node of the hidden layer is connected to each node of the output layer. Mathematically, each hidden layer element is equal to the sum of the product of the inputs and their assigned weights. The output of each hidden layer element is obtained by applying a function (f_j) to the sum (i.e. the input of the element). As in the case of the input layer, this output, in turn, is weighted (w_{jk}) and sent as inputs to each of the output elements. The final output, i.e. each output element, y_k , is obtained by applying a function (f_k) to the sum of the inputs from the hidden layer. Therefore (Ripley, 1996), :

$$y_k = f_k \left(\sum_{i \rightarrow j} w_{jk} f_j \left(\sum_{i \rightarrow j} w_{ij} x_i \right) \right) \quad (3.1)$$

In the present case, as there is only one output, the generic output will be referred to as y rather than y_k .

The *activation* functions $f_j(\bullet)$ and $f_k(\bullet)$ are typically sigmoidal or logsig as shown below:

$$f_x = \frac{1}{1 + e^{-x}} \quad (3.2)$$

However, different activation functions can be used to detect different features, such as Gaussian (Equation 3.3) for learning the middle, while Gaussian-complement (Equation 3.4) for learning the extremas. Another popular activation function is $\tanh$ (Equation 3.5), which has many benefits: mean

of zero, variance less than 1 and the derivative neither magnifying nor attenuating the backpropagation errors.

$$f_x = e^{-x^2} \quad (3.3)$$

$$f_x = 1 - e^{-x^2} \quad (3.4)$$

$$f_x = \frac{e^{+x} - e^{-x}}{e^{+x} + e^{-x}} \quad (3.5)$$

3.5.3. Learning

From above, it becomes apparent that the obtained output depends on the weights of the interconnections. Therefore, it is paramount that these weights be determined correctly so that they are representative of the property being estimated. The process of determination of weights is called “learning” or “training”.

Learning is a process by which the neural network adjusts its weights to reflect changes in the environment, i.e. during learning, it assimilates the patterns. The sample data set for the property being estimated is used for this purpose. Assume that there are 100 sample points. The network starts learning with sample point number one. It begins with arbitrary weights; typically, all weights are set to random numbers with a mean of zero and a standard deviation of one. In most cases, it doesn't really matter what the initial weights are. Using the initial weights, the network makes a prediction, i.e. it computes the output y as described above. It then compares the output with the sample and determines the error. Next, it adjusts the weights (presented in the next sub-section)

based on their contribution to the determined error. This process of propagating the effect of the error onto all the weights is called *back propagation*. Using the adjusted weights, the network then computes an output for sample number two. The weights are adjusted once again to reflect the error. Ultimately, it goes through all the samples, adjusting the weights on each occasion. When a neural network is done training on a sample set, it is said to have completed an *epoch*.

Typically, the weights after an epoch are still unsatisfactory. Satisfaction is measured by several criteria such as the magnitude of error made during an epoch. The level of error that is satisfactory is set by the user depending on the task. To make the network satisfactory, the learning/training process is restarted with the initial weights being the final weights of the previous epoch. When the error for an epoch is deemed satisfactory, the network is “trained”.

Various architectural details affect neural network performance (ability to be trained and predict within required limits of error). General rules exist on the appropriate architecture to use for a given problem. For example, as each neuron in a hidden layer absorbs a particular feature of the relationship between the inputs and outputs, a very complex relationship will require a large number of neurons. When combined, as summed inputs to another hidden layer, a set of neurons can absorb even more features. It has been proved that two hidden layers with the appropriate number of neurons (typically no more than twice the number of inputs) can approximate any mathematical function.

3.5.4. Back propagation

There are various back propagation algorithms available in the literature. In this section, we detail one of the simpler algorithms. However, this algorithm contains the basic idea used by all back

propagation algorithms. Therefore, this section will give readers the core idea behind back propagation technique without subjecting them to mathematical concepts (that are difficult to read) used by the various other algorithms.

For correcting the weight w_j connecting the hidden layer node j to the output, assume that the error gradient (a measure of the contribution of the node towards the total error) for that node (e_j) was known. The correction in the weights should then be proportional to this measure. If w_j is the weight connecting the j^{th} node to the output, the corrected weight is given by:

$$w_j (\text{new}) = w_j (\text{old}) - \Delta w_j \quad (3.6)$$

where, the correction Δw_j is given by:

$$\Delta w_j = e_j \cdot I_j \quad (3.7)$$

where, I_j is the total input to the node j . e_j being the gradient, this method is known as the gradient descent method. A learning rate (lr) is incorporated to control the amount of correction. This is shown below:

$$w_j (\text{new}) = w_j (\text{old}) - lr \cdot e_j \cdot I_j \quad (3.8)$$

e_j , for a layer s , is obtained from the equation (derivation omitted as it is an established equation) (Neuralware, 1996):

$$e_j^s = f'(I_j^s) \cdot \sum_k (e_k^{s+1} \cdot w_{kj}^{s+1}) \quad (3.9)$$

Starting with the output layer, the equations above are used iteratively to correct all weights.

The advantage in using the sigmoidal function in (3.2) is that its derivative is easily computed (making programming easier):

$$f'(\bullet) = f(\bullet) \cdot (1 - f(\bullet)) \quad (3.10)$$

Therefore, (3.9) reduces to:

$$e_j^s = I_j^s \cdot (1 - I_j^s) \cdot \sum_k (e_k^{s+1} \cdot w_{kj}^{s+1}) \quad (3.11)$$

3.5.5. Quick Stop Training

One problem with neural networks is not knowing when to stop training. At the initial stages of training, the neural network learns more and more about the relationship between the inputs and the output. As training continues, this understanding improves further. However, if training is not stopped, then the network starts memorizing the data, i.e. in order to improve training performance, it begins to include the noise component into the “understanding” in an effort to further drive down the training error. In other words, the network starts moving away from the general relationship (stops generalizing). This phenomenon is called over training. Ideally, neural network training should be stopped as soon as the network stops generalizing.

To prevent over training, one could use the quick stop training method. This method helps identify the point at which generalization stops. In this method, a small subset of the data is isolated from the training subset. After each epoch of training, the error on this subset (calibration subset) is noted. Note that the calibration subset is not used in training in any way, i.e. the calibration error is not backpropagated. Initially, both training and calibration errors go down. When the network stops generalizing, the calibration error stops decreasing with continued training even though the training error keeps reducing. The network for which the calibration error was the lowest is considered the fitted or trained model.

3.5.6. Implementation of the trained neural network

Once the network is trained, the weights are no longer modified, i.e. network is available for prediction. The various inputs are fed into the network and predictions made based on the final weights. Prediction using the weights is a trivial exercise requiring no more than a spreadsheet.

Models are typically validated prior to being considered final.

IV. NEURAL NETWORK MODELING

The neural network modeling software Neuroshell2 was used for the purpose. All models had two inputs (Am and Cs counts) and one output (percentage ash). The details of the neural network modeling are given in the next few sections.

4.1. Performance Indices

The performance of the models were based on the R^2 and bias of the predictions, and the mean absolute error (MAE). The R^2 is obtained from the equation:

$$\begin{aligned} R^2 &= 1 - \frac{SSE}{SS_{yy}} \\ SSE &= \sum (y - \hat{y})^2, \\ SS_{yy} &= \sum (y - \bar{y})^2, \end{aligned} \tag{4-1}$$

where y is the true value, $\hat{y}$ is the predicted value and $\bar{y}$ is the mean of the true values. The bias is the difference between the means of the true values and the predicted values.

4.2. Model Performances for Kohonen-Based-Data-Split Data (Yu, 2003)

Please see Appendix – I for the data used in the various models and the model parameters.

4.2.1. Screened Data

The model (called S1) with the best fit for this data consisted of a single hidden layer (see Figure 3.3) with 22 neurons. The neurons used the logsig activation function (see Section 3.5.2). The model performance in training, calibration and prediction subsets is given below:

Table 4.1. Performance of the neural network on screened data (Kohonen)

Subset	R Square	MAE	Bias
Training	0.32	1.52	0.2
Calibration	0.55	0.76	0.38
Prediction	0.51	1.14	0.09

Performance is evaluated based on the prediction set performance. However, for further insight, the training and calibration predictions are also given. As can be seen, the bias of the prediction set (0.09) is very low, which makes for a good predictor (none of the biases are high). Since the prediction set contains seven samples, this implies that the neural network predictions were accurate (insignificant bias) at about 2-3 minutes worth of data. This is because each sample represents 20 seconds of coal. The R^2 of predictions, however, could be better.

While analyzing network performance, it is also important to look at the training and calibration performances, as in subsequent real life implementation of the network, it is likely that an incoming value is similar to those in the training and calibration subsets. Both biases are low, which implies that on average, the customer can expect very good performance from the neural network.

4.2.2. Unscreened Data

The model with the best performance was the jump net J3 (Figure 4.1). There are three hidden layer slabs (HL Slabs). The number inside the slabs indicate the number of neurons. The activation functions for the HL Slabs 1, 2 and 3 were all logsig (see Section 3.5.2).

Figure 4.1. Jump net J3

The performance of the model is given in Table 4.2.

Table 4.2. Performance of the neural network on unscreened data (Kohonen)

Subset	R Square	MAE	Bias
Training	0.55	1.51	0.2
Calibration	0.19	1.15	0.11
Prediction	0.25	1.75	0.07

The performance of the neural network is worse than for screened data in terms of R^2 . However, as before, the biases are still very small, which implies that the neural network predictions are accurate at 2-3 minute level.

4.2.3: Combined Data

The model with the best performance was S3 (Figure 4.2). Each hidden layer had 22 neurons. All activation functions were logsig (see Section 3.5.2).

Figure 4.2. The neural net S3

The performance is given in Table 4.3.

Table 4.3. Performance of the neural network on combined data (Kohonen)

Subset	R Square	MAE	Bias
Training	0.41	1.45	0.03
Calibration	0.44	2	0
Prediction	0.3	1.82	0.26

The biases are still very small, though R^2 , as before, is poor. Since the prediction set contains 24 samples, the accuracy is at the 8 minute level rather than the 2-3 minute interval as before. Notice the slight increase in MAE. This is to be expected given the heterogeneity of the combined data. The performance of the combined model is acceptable given its low bias in all the subsets.

4.2.4: Combined Modeling vs. Separate Modeling

This section examines whether there was any advantage to modeling the sample data separately as screened and unscreened. If a mine operator chose to deploy two separate neural network models for their screened and unscreened data, then the overall performance of the neural network method of calibration would be the cumulative performance of the two models. In other words, the predictions would have to be merged into one group, and their overall R^2 and bias would have to be obtained. This was the approach taken in comparing the performance of the two separate models with that of the single combined data model. Tables 4.4 through 4.6 give the “pooled” performance which was obtained by merging the prediction subsets of the screened and unscreened models.

Table 4.4. Performance comparison of the pooled performance in training subset (Kohonen)

Subset	R Square	MAE	Bias
Screened	0.32	1.52	0.2
Unscreened	0.48	1.47	0.01
Pooled	0.5	1.47	0.17
Combined	0.41	1.45	0.03

Table 4.5. Performance comparison of the pooled performance in calibration subset (Kohonen)

Subset	R Square	MAE	Bias
Screened	0.55	0.76	0.38
Unscreened	0.15	1.86	0.37
Pooled	0.31	1.33	0.07
Combined	0.44	2	0

Table 4.6. Performance comparison of the pooled performance in prediction subset (Kohonen)

Subset	R Square	MAE	Bias
Screened	0.51	1.14	0.09
Unscreened	0.25	1.75	0.07
Pooled	0.22	1.64	0.28
Combined	0.3	1.82	0.26

From the above tables, it is apparent that the pooled performance is not any better than the combined model.

4.3. Model Performances for GA-Based-Data-Split Data

Please see Appendix – II for the data used in the various models and the model parameters.

Note that the training, calibration and prediction subsets in this section are different from those in the previous section. Therefore, a one-to-one comparison is not possible.

4.3.1: Screened Data

The model with the best performance was J1 (Figure 4.3). It has one hidden layer slab with a single neuron and activation function tanh (see Section 3.5.2).

Figure 4.3. Jump net J1 (tanh)

The performance of the model is given in Table 4.5

Table 4.5. Performance of the neural network on screened data (GA)

Subset	R Square	MAE	Bias
Training	0.23	1.68	-0.59
Calibration	0.52	1.66	-0.08
Prediction	0.16	1.74	-0.66

The biases of the various subsets are small. As before, however, the R^2 's are also low. The calibration R^2 is higher than for the training and prediction subsets.

4.3.2: Unscreened Data

The best architecture for this was S2, the two hidden slab network (Figure 4.4). Each hidden slab had 4 neurons with the tanh activation function (see Section 3.5.2). The performance of the model is given in Table 4.6.

Figure 4.4. The architecture S2 (tanh)

Table 4.6. Performance of the neural network on unscreened data (GA)

Subset	R Square	MAE	Bias
Training	0.24	1.89	0.10
Calibration	0.17	1.93	0.19
Prediction	0.30	2.11	0.10

Like the models on other data, this model is also characterized by low bias and R^2 .

4.3.3: Combined Data

The best model for this data is given in Figure 4.5. It's a single hidden slab network with 9 neurons and tanh activation function (see Section 3.5.2). The performance of the model is given in Table 4.7.

Figure 4.5. Jump net J1 (tanh)

Table 4.6. Performance of the neural network on combined data (GA)

Subset	R Square	MAE	Bias
Training	0.37	1.55	-0.40
Calibration	0.37	1.67	-0.10
Prediction	0.34	2.3	-0.50

Like the models on other data, this model is also characterized by low bias and R^2 . The performance of the model seems uniform across the subsets in terms of R^2 .

The MAE of the prediction set is higher, but that is in line with the heterogeneity of the combined data.

4.3.4: Combined Modeling vs. Separate Modeling for GA based Data Split

As in Section 4.2.4, in order to explore the relative advantage of modeling the data separately as screened and unscreened, the prediction sets of the unscreened and screened models were merged to obtain the “pooled” performance (Tables 4.7 through 4.9).

Table 4.7. Performance comparison in training subset (GA based split)

Subset	R Square	MAE	Bias
Screened	0.23	1.68	-0.59
Unscreened	0.24	1.89	0.10
Pooled	0.25	1.80	-0.20
Combined	0.37	1.55	-0.40

Table 4.8. Performance comparison in calibration subset (GA based split)

Subset	R Square	MAE	Bias
Screened	0.52	1.66	-0.08
Unscreened	0.17	1.93	0.19
Pooled	0.21	1.80	0.06
Combined	0.37	1.67	-0.10

Table 4.8. Performance comparison in prediction subset (GA based split)

Subset	R Square	MAE	Bias
Screened	0.16	1.74	-0.66
Unscreened	0.30	2.11	0.10
Pooled	0.18	1.94	-0.25
Combined	0.34	2.3	-0.50

The combined data model outperformed the pooled performance in terms of R^2 in all the three subsets. Their performances in terms of bias and MAE are similar. While the combined data model's R^2 is impressive when compared to the pooled performance, the benefits need not be as spectacular from a practical point of view. This is because not much is gained from the higher R^2 value of the combined model; the R^2 in an absolute sense is still fairly low, while the MAE is still fairly high, making individual predictions still very unreliable though the average of predictions will be close to the true value (as will be for the pooled performance as well). Therefore, the two performances can be deemed practically similar.

4.4. Conclusions on Modeling

From the above, it becomes apparent that no matter how the data is split and modeled, neural networks always predicted the ash content with low bias. There was no advantage to modeling the data separately as screened and unscreened. Therefore, it is recommended that the mine use the combined data model as their neural network model. Of the two combined data models (Kohonen and GA), it is recommended that the GA based combined data model be used since the data subdivisions were better with GA.

V. PROGRAMMING

Custom software had to be developed twice during the project (see Appendix III). Initially, a program was written to pre-process the data (OLA output which were saved in the form of a text file) to obtain the 40 relevant Am and Cs readings for each sample. After neural network model development, this code was enhanced so that it could process OLA readings and apply the developed neural network model to make predictions.

All code was written by Ganguli in C++ and implemented in DOS.

5.1. Pre-processing

Figure 5.1 shows a typical OLA output. Each row represents 0.5 seconds of data. The important columns in each row are marked.

□□00	9.6512	10.541	10224	8271	0.0	49.0000	-118.500	7E
□□00	10.516	11.431	9521	6899	0.0	49.0000	-118.500	79
□□00	10.268	12.813	8524	5411	0.0	49.0000	-118.500	6F
□□00	9.2344	13.385	8143	5064	0.0	49.0000	-118.500	7A
□□00	10.908	13.195	8268	4556	0.0	49.0000	-118.500	82
□□00	10.779	14.410	7502	3806	0.0	49.0000	-118.500	72
□□00	9.3237	14.319	7557	4253	0.0	49.0000	-118.500	81
□□00	8.6513	13.608	7999	4844	0.0	49.0000	-118.500	90
□□00	6.6326	12.378	8826	6915	0.0	49.0000	-118.500	8A
□□00	6.6326	12.378	10411	11773	0.0	49.0000	-118.500	97
□□00	7.6071	9.5043	11108	10739	0.0	49.0000	-118.500	9A
□□00	9.9832	10.302	10420	8320	0.0	49.0000	-118.500	7A
□□00	9.5672	12.090	9032	6425	0.0	49.0000	-118.500	79
□□00	8.5493	12.810	8526	5780	0.0	49.0000	-118.500	83
□□00	7.7074	13.117	8319	5490	0.0	49.0000	-118.500	7E

Figure 5.1. Sample OLA output

Among other things, the output consists of the Am and Cs counts as well as an estimate of the weight scanned by the OLA in the previous 0.5 seconds. Each sample consisted of 40 lines (for 20 seconds of data). The software suite (consisting of 2 separate programs) extracted the three columns and used the weight estimate to assign weights to the Am and Cs readings for that row. The average of the 40 weighted Am and Cs readings constituted the Am and Cs reading for that sample.

The code for the software suite is given in the Appendix - III. The program `crop.cpp` read the raw data file for each sample, conducted the weighting and wrote the results to a separate file (for each sample). The program `gen_data_file.cpp` then read each file automatically (in one sweep), compiled the information and wrote a single data file that contained the Am and Cs averages for the 104 samples. The corresponding laboratory ash values were subsequently added to the file to make it the final input file for neural network modeling.

5.2. Application of Neural Network Model

The essentials from `crop.cpp` were modified to obtain `ucm.cpp`, a program that could not only read OLA raw data files, but could also apply the fitted neural network model. The program had to adapt to the fact that during implementation the OLA data file would be thousands of lines long. Since the neural network modeling was built on Am and Cs values that were averages of 40 lines, the program read and averaged 40 lines to obtain the average Am and Cs values. These averages were sent to the neural network function to obtain the ash prediction (for the 20 seconds representing the 40 lines). This continued until the data file was completely read.

The neural network function was obtained by using a feature of the modeling software Neuroshell2. Neuroshell2 has the ability to output C++ code representing the fitted model. This outputted code was slightly modified to allow seamless interaction with ucm.cpp.

VI: CONCLUSIONS AND RECOMMENDATIONS

The following is a summary of the project:

- Neural networks were used to calibrate an online analyzer
- 104 samples were collected from different parts of the mine over a period of several months
 - 47 of the 104 samples were from screened coal, while the rest were from unscreened coal.
 - Each sample amounted to 20 seconds of coal on a running conveyor belt (approximately 3 tons).
 - Each sample was sub-sampled to determine its ash content. The sub-samples were individually analyzed for twenty samples to obtain an estimate of the variance within each sample.
- Data was split into three “equivalent” subsets to ensure better network performance evaluation
 - In a novel approach, genetic algorithms were used to split the data into three “similar” groups
 - Equivalent subsets were also obtained by a second approach where each subset was built from the various data clusters or groups (as determined by Kohonen network) within the dataset.
- Relative advantage of modeling the screened and unscreened data separately was explored.

The following can be concluded based on the project:

- Neural networks (independent of data division method) predicted the ash content with insignificant bias at the 2-3 minute level. In other words, the ash predictions for 2-3 minute intervals were close to true values.
- Individual ash predictions (each for 20 seconds of coal) were not very reliable.
- There was no advantage to modeling the screened and unscreened coal separately

Based on the project, the following recommendations are made:

- Poor performance was primarily due to noise in the data. Therefore, noise reduction should be the main goal for future analyzer applications. This can be done in two ways:
 - Increase reliability of analyzer readings. Better technology is needed so that the analyzer reading is not based on a narrow stream of gamma rays that does not sample the entire width of the belt. At the very least, another set of gamma sources (and associated counters) that take a horizontal sample of the belt, may improve analyzer readings.
 - Increase sample length for calibration. However, increasing it to a large period (such as 30 minutes), will not allow the analyzer to be used for real-time control since real-time control requires analyzer readings that are accurate even at short durations.

ACKNOWLEDGMENTS

The authors would like to acknowledge the funding from the U.S. Department of Energy that made this project possible. The support of the project partners, Usibelli Coal Mine and Golden Valley Electric Association is greatly appreciated. Special mention is due to Alan Renshaw, Fred Walis, Alan McCaleb, Vada Peterman, Steve Denton, Dan Graham and others at the Usibelli coal mine for their strong support of the project. The support of the University of Alaska Fairbanks is also recognized.

REFERENCES

- Bowden, G. J., Maier, H. R. and Dandy, G.C., 2002, "Optimal Division of Data for Neural Network Models in Water Resources Application," **Water Resources Research**, Vol. 38, No. 2, 10.1029/2001WR000266, 11 pp.
- Hines, W. W. and Montgomery, D.C., 1980, **Probability and Statistics in Engineering and Management Science**, John Wiley and Sons, ISBN 0-471-04759-7, pp. 634.
- Ganguli, R., 2001, "A Critical Review of Analyzers," **Mineral Resources Engineering**, Vol. 10, No. 4 (December), pp. 435-445.
- Goldberg, D.E., 1999, **Genetic Algorithms in Search, Optimization and Machine Learning**, Addison-Wesley, International Student Edition.
- Kamada, H., Kawaguchi, H. and Onodera, J., 1986, "On the Coal Blending Process Control by On-line Ash Monitors," **Proceedings** of the 10th International Coal Preparation Congress, Edmonton, Alberta, Canada, September, pp. 245-266.
- Kohonen, T., **Self-Organization and Associative Memory**, Springer-Verlag, New York, 1984.
- Ripley, B. D., 1996, **Pattern Recognition And Neural Networks**, Cambridge University Press, pp. 403.

Yu, S., 2003, **Calibration of an Online Ash Analyzer Using Neural Networks**, M.S. (Mining Engineering) Thesis, University of Alaska Fairbanks, August.

Yu, S., Ganguli, R., Walsh, D.E., Bandopadhyay, S. and Patil, S.L., 2003, “Calibration of On-line Analyzers Using Neural Networks,” Accepted, **Mining Engineering**.

Appendix – I: Data and Models for Kohonen-based-Data-Division Models

Explanation of Categories

T: Training, P: Calibration, V: Validation

Table A1-1: The Data of Network S1 for Screened Data

Subgroup	Ash	Am-241	Cs-137	Category
1	12.1	16727	30487	T
1	9.51	19042	42837	T
1	11.7	20186	48606	T
2	13.4	15135	22743	T
2	11.1	15171	23441	T
2	14.4	15230	23505	T
2	10.2	15235	24015	T
2	11.1	15358	24145	T
2	13.7	15416	23875	T
2	9.37	15462	25490	T
2	9.95	15540	24991	T
2	10.2	15751	26726	T
2	11.6	15981	29029	T
2	9.81	16475	29566	T
3	13.6	13743	17992	T
3	14.5	13973	18639	T
3	16	14042	19242	T
3	13.3	14255	21073	T
3	11.2	14273	20120	T
3	9.4	14310	20176	T
3	12.8	14385	20186	T
3	12.4	14525	20543	T
3	12	14553	21171	T
3	13.2	14628	21238	T
3	10.9	14705	22195	T
3	21.2	14847	21690	T
4	10.7	8524	5985	T
4	9.55	12345	13854	T
4	9.8	12513	16159	T
4	15.7	12732	14413	T
4	13.6	12763	14432	T
4	9.74	13344	16895	T
4	12.9	13514	17105	T
1	14.3	19965	47322	P
2	14.2	15171	23321	P
2	9.96	15929	26954	P

3	9.71	13984	19305	P
3	13.5	14550	20932	P
4	9.86	12058	13163	P
4	13.9	13430	16627	P
1	11	20151	47894	V
2	13	15289	23738	V
2	11.2	16282	28447	V
3	10.2	13744	18118	V
3	16	14692	21166	V
4	10.6	13163	16393	V
4	15.7	13154	16276	V

Weights (architecture) of S1

Figure A1-1. The Selected ANN for Screened Data (S1)

Table A1-2: The Weights of Selected Network S1 for Screened Data

Link 1				Link 2	
Neurons	Bias	Am-241	Cs-137	Bias	
1	-0.74589	-0.67519	-0.89872	1	-0.20365
2	-0.27783	-0.61318	-0.03941	2	-0.20545
3	5.973482	5.862846	0.613737	3	-0.53434
4	0.631881	0.373364	0.763532	4	0.051431
5	1.246734	1.214726	2.033576	5	0.068694
6	1.246734	1.214726	2.033576	6	0.002023
7	-0.45871	-0.40393	-1.09657	7	-0.07822
8	-0.53646	-0.38558	-0.75454	8	-0.24597
9	4.220655	4.047376	0.722245	9	0.063127
10	-2.67192	-3.06588	-0.62426	10	-0.11905
11	-2.75816	-2.8083	-0.62252	11	-0.09441
12	2.670087	2.836096	0.989235	12	0.186379
13	5.481342	5.177249	0.041882	13	0.083384
14	-0.30576	-0.69783	-1.06261	14	-0.30422
15	-4.72464	-4.66815	1.300338	15	-0.18545
16	-0.56424	-0.27665	-0.71825	16	-0.13214
17	6.113766	6.58634	-0.17933	17	-0.077726
18	-4.49419	-4.23711	1.16721	18	-0.68462
19	2.657147	2.619646	20.58695	19	-0.99519
20	-0.65825	-0.22105	-0.81601	20	-0.14648
21	-0.6471	-0.31063	-1.18861	21	-0.27825
22	-2.57251	-2.59208	-1.21706	22	-0.23412
	-0.47922	-0.19121	-0.67106		-0.10819

Table A1-3: The Data of Network J3 for Unscreened Data

Explanation of Categories

T: Training, P: Calibration, V: Validation

Subgroup	Ash	Am-241	Cs-137	Category
7	13.6	20400.3	49258.1	T
1	14.1	6719.72	2932.87	T
1	12.2	8046.34	5313.38	T
1	15.2	7009.89	3183.71	T
5	13.9	12471.4	14374.9	T
5	18.8	11461.9	12061.5	T
1	13.3	8103.06	4674.93	T
1	13.8	1159.5	11601.6	T
1	17.8	7232.95	3445.96	T
2	10.7	8524.41	5985.44	T
2	10.9	9020.11	6822.99	T
2	7.48	8624.6	6457.92	T
2	10.9	8356.3	5542.83	T
2	13.8	8537.01	6153.13	T
2	14.6	8324.6	5384.55	T
2	11.7	8749.08	5869.06	T
2	12.9	8651.92	5885.65	T
3	11.7	9320.16	7136.71	T
3	16.8	9797.84	8890.44	T
5	12.7	11553.7	14668.3	T
5	12.4	11346.3	11929.1	T
5	13.6	11856.1	12576.6	T
3	15.4	9709.09	8324.05	T
3	11.7	9323.1	7049.56	T
3	12.2	9827.68	7920.14	T
5	11.3	11786.2	13331	T
3	12.9	10198.6	8593.87	T
3	12.3	9182.41	7024.49	T
3	11.9	10161.4	8343.25	T
4	11.5	10593.5	9632.77	T
4	10.3	10983.2	10468.2	T
4	11.7	10460.2	9674.78	T
4	13.9	10963.6	11033.3	T
4	11.1	10887.2	10412.7	T
4	18.3	10870.7	10327.6	T
4	14.6	10444.7	9082.16	T

5	11.7	12016.1	12866.3	T
5	14.2	11567.5	11806.3	T
6	8.9	12431.3	15552.4	T
6	18	12890.8	15582.9	T
6	12.3	13364.4	18158.6	T
7	15.5	14398.4	20769.7	T
7	9.4	16286.6	29106.8	T
1	23.3	6294.94	2410.12	P
2	10.3	8081.35	5174.37	P
3	11.6	9263.38	7718.51	P
4	8.74	10021.4	10953.6	P
4	15.4	10953	9811	P
5	14	11856.7	12869.2	P
6	11.3	13216.8	16078.4	P
7	9.75	15398.9	24962.8	P
1	13.5	7185.13	3358.75	V
2	11.5	9154.27	6781.35	V
3	11.6	9245.48	6846.6	V
4	9.75	10398.8	10593.4	V
4	13.5	10774.7	11252.9	V
5	18	11265.6	11317.5	V
6	15.4	12837.8	15867.5	V

Figure A1-2. The Selected ANN for Unscreened Data (J3)

Table A1-4: The Weights of Selected Network J3 for Unscreened Data

		Link 1	
Neurons	Bias	1	2
1	64.57452	64.78395	61.52681
2	-1.95702	-1.75759	1.413482
3	81.42535	81.80307	46.1113
4	-7.65767	-7.69496	-8.89342
5	24.72468	24.47814	18.03403
6	-22.1405	-21.913	-12.7077

		Link 2					
Neurons	Bias	1	2	3	4	5	6
1	-1.11636	-1.1994	-0.26405	0.196716	0.226338	-0.24307	0.077406
2	65.05728	65.18447	-7.37665	-1.00553	-7.20698	-0.67354	-91.9276
3	-1.07063	-1.34474	53.89879	0.736303	-2.6248	1.293879	4.359657
4	13.68514	13.76044	-15.5164	-17.1655	-13.7562	-6.25383	-19.9908
5	-5.4445	-5.15489	-0.1734	0.342407	-3.28387	5.998775	6.377634
6	-12.3099	-11.9484	7.286476	3.593075	17.91539	27.74553	-33.9822

		Link 3					
Neurons	Bias	1	2	3	4	5	6
1	-2.66643	-2.72642	-0.10443	-41.6843	-2.00534	18.84122	-0.16918
2	0.101638	0.209802	0.595594	-0.20619	-0.08229	1.066423	0.218528
3	4.032801	3.979224	-0.09423	8.575474	3.873367	4.11087	1.046518
4	4.462534	4.843514	-47.1152	-1.39018	-91.0768	1.042263	-1.54726
5	-0.29322	0.204076	-0.05364	2.447582	0.108723	-0.21471	-0.19265
6	0.27426	0.158882	0.261914	-1.80691	0.062965	0.12614	0.160387

Link 4

Neurons	Bias	1	2	3	4	5	6
1	-0.15096	0.307165	0.176097	-1.75667	2.877212	-0.23533	0.304382

Link 5

Neurons	Bias	1	2
1	0.524038	0.220862	-0.04675
2	63.65193	64.05031	43.29039
3	-0.91303	-1.15422	3.538816
4	41.89532	41.49565	16.18946
5	6.080294	6.033417	0.120452
6	-23.1227	-22.7983	-0.95595

Link 6

Neurons	Bias	1	2
1	23.88237	23.99498	2.31733
2	-28.9145	-28.7326	-31.2572
3	-17.7872	-17.945	0.209599
4	6.011293	6.080088	1.116019
5	-1.69482	-1.16305	-0.0969
6	0.951287	0.970807	0.395297

Link 7

Neurons	Bias	1	2
1	1.277747	1.704396	-0.15237

Link 8

Neurons	Bias	1	2	3	4	5	6
1	-5.60473	-5.7477	-5.04504	-6.25829	-9.00916	-3.45414	19.37949
2	-20.466	-20.2723	-0.17088	2.02009	4.168836	1.428088	43.38866
3	-0.94423	-1.04535	1.501835	-0.17177	1.513922	0.950036	-27.8883
4	-7.79999	-7.66	0.563622	24.97722	6.169506	-24.0109	3.268994
5	0.310234	0.150122	0.105459	-0.13945	0.16726	-0.28979	-0.14111
6	-0.16828	-0.37657	-0.19135	0.294431	0.370592	-0.2713	-0.79382

Link 9

Neurons	Bias	1	2	3	4	5	6
1	-0.47214	-0.2677	0.14132	0.14045	-0.23151	-0.32328	0.116373

Link 10

Neurons	Bias	1	2	3	4	5	6
1	-0.15486	0.141709	-0.09363	-2.02723	-0.3912	0.082245	-0.09039

Table A1-5: The Data of Network S3 for Combined Data

Explanation of Categories

T: Training, P: Calibration, V: Validation

Am-241	Cs-137	Ash	Category
16287	29107	9.4	T
15399	24963	9.7	T
16727	30487	12.1	T
19042	42837	9.5	T
13973	18639	14.5	T
20151	47894	11	T
20400	49258	13.7	T
15171	23441	11.1	T
15230	23505	14.4	T
15462	25490	9.4	T
15416	23875	13.7	T
15235	24015	10.2	T
12431	15552	8.9	T
12891	15583	18	T
12838	15868	15.4	T
13217	16078	11.3	T
15929	26954	10	T
15981	29029	11.6	T
13743	17992	13.6	T
14692	21166	16	T
14553	21171	12	T
13984	19305	9.7	T
14273	20120	11.2	T
14310	20176	9.4	T
14525	20543	12.4	T
13514	17105	12.9	T
10953	9811	15.4	T
10887	10413	11.1	T
10983	10468	10.3	T
14847	21690	21.2	T
12471	14375	13.9	T
12732	14413	15.7	T
10775	11253	13.5	T
11266	11318	18	T
11568	11806	14.2	T
11346	11929	12.4	T
11856	12577	13.6	T
12513	16159	9.8	T
8081	5174	10.3	T

13163	16393	10.7	T
9828	7920	12.2	T
9709	8324	15.4	T
9323	7050	11.7	T
9798	8890	16.8	T
10445	9082	14.6	T
10594	9633	11.5	T
10460	9675	11.7	T
7010	3184	15.2	T
7185	3359	13.5	T
7233	3446	17.8	T
9020	6823	10.9	T
1160	11602	13.8	T
8652	5886	12.9	T
12058	13163	9.9	T
11786	13331	11.3	T
9154	6781	11.5	T
9245	6847	11.6	T
8325	5385	14.6	T
8356	5543	10.9	T
8749	5869	11.7	T
16475	29566	9.8	P
19965	47322	14.3	P
15358	24145	11	P
15540	24991	10	P
15289	23738	13	P
14042	19242	16	P
14628	21238	13.2	P
14398	20770	15.5	P
13430	16627	13.9	P
12763	14432	13.6	P
13154	16276	15.7	P
10964	11033	13.9	P
12016	12866	11.7	P
11857	12869	14	P
9320	7137	11.7	P
9263	7719	11.6	P
10161	8343	11.9	P
8524	5985	10.7	P
8103	4675	13.3	P
8046	5313	12.2	P
8625	6458	7.5	V
13344	16895	9.7	V
13744	18118	10.2	V
14255	21073	13.3	V
10399	10593	9.8	V

8537	6153	13.8	V
20186	48606	11.7	V
10021	10954	8.7	V
6295	2410	23.3	V
14550	20932	13.5	V
15171	23321	14.2	V
10871	10328	18.3	V
15135	22743	13.4	V
12345	13854	9.6	V
6720	2933	14.1	V
11554	14668	12.7	V
10199	8594	12.9	V
14705	22195	10.9	V
9182	7024	12.3	V
11462	12062	18.8	V
13364	18159	12.3	V
15751	26726	10.2	V
14385	20186	12.8	V
16282	28447	11.3	V

Weights (architecture) of S3

Figure A1-3. The Selected ANN for Combined Data (S3)

Table A1-6: The Weights of Selected Network S3 for Combined Data

Link 1			
Neurons	Bias	1	2
1	-73.8555	-73.7848	-48.5554
2	-60.484	-60.8194	-31.8186
3	5.823186	5.712551	-10.2996
4	-41.721	-41.9795	-12.5035
5	-9.06778	-9.09979	0.019516
6	9.26161	9.316398	12.354
7	-15.1454	-14.9945	-0.36839
8	8.029562	7.856286	-4.84619
9	-11.9016	-12.2955	-14.4947
10	1.584448	1.534312	-2.50335
11	3.217983	3.383992	47.74329
12	8.426618	8.122525	1.840759
13	-13.5519	-13.9439	1.41739
14	1.715699	1.772188	-0.61746
15	-20.8603	-20.5727	-1.68383
16	10.67848	11.15105	-7.75757
17	-11.8964	-11.6393	0.826889
18	-24.4503	-24.4879	-13.6419
19	10.45545	10.89264	0.984817
20	8.874375	9.210843	-5.16448
21	-17.1542	-17.1738	0.134069
22	-17.4843	-17.1963	-2.08168

Link 2

Neurons	Bias	1	2	3	4	5	6	7
1	-8.67267	-8.58311	2.944627	0.217786	2.46611	0.641551	0.155486	3.790001
2	5.794791	6.11278	-12.279	-1.33692	-1.51762	-1.07939	-7.00451	7.458334
3	-40.7907	-40.8332	-29.7242	1.834587	-12.0336	-31.0243	1.969544	-16.2685
4	-12.1236	-11.9443	1.02618	-0.22476	0.435418	7.192913	-0.11879	4.406643
5	-16.6486	-16.7003	-0.52884	0.28057	0.213312	-7.48625	0.069025	3.657493
6	-16.5724	-16.8009	-1.6528	0.256238	0.090427	0.016663	0.029172	3.808537
7	-12.5014	-12.691	1.143177	0.037272	2.500492	0.18079	0.109964	4.55024
8	-14.7485	-14.4774	-0.12256	-0.00509	5.375339	2.503869	3.993878	5.75362
9	-14.2052	-14.3563	43.82904	-0.02032	20.69882	10.02642	-0.15298	-5.76611
10	-0.10624	-0.3492	0.5685	-1.52762	-0.53576	-1.30022	-1.23918	0.586056
11	-0.65903	-0.51395	0.346274	-0.32323	-0.86647	-1.08438	-0.27517	5.720232
12	-6.89545	-7.16206	2.604828	-0.07631	0.46357	-0.13348	0.314727	4.317586
13	-32.6741	-33.0322	-2.70313	1.75535	12.10939	4.219033	3.139835	-3.25003
14	22.40381	22.90859	-7.32677	-2.99888	-1.42668	-1.81957	-2.25851	-2.08138
15	-9.45224	-9.72673	-0.04913	0.174227	-0.30582	-0.49363	-0.22016	9.810906
16	-16.4939	-16.182	2.50119	-0.44852	1.033364	-0.45223	-0.00767	9.481365
17	25.55306	25.52779	-7.48038	-4.61735	-23.1858	-28.6602	-28.1713	-8.69675
18	-13.4628	-13.6463	3.362736	0.056267	3.426343	0.395517	0.318058	9.851123
19	0.100191	0.012682	-0.00436	-0.52675	0.03942	-0.09342	-0.58536	0.354513
20	-12.0736	-12.2481	2.842413	0.164899	1.979122	-0.3634	0.097392	0.84206
21	-42.9844	-42.4954	-1.66626	2.469472	11.32362	19.14831	0.312622	-5.9508
22	-72.588	-72.3131	0.152221	0.218651	0.128074	-1.31523	0.036502	3.230156
Neurons	8	9	10	11	12	13	14	15
1	0.251089	30.21097	0.009776	0.1919	0.355193	1.024936	0.186145	0.173171
2	-1.19166	3.773919	-1.54251	1.31147	-1.44587	-1.37313	-1.59164	-0.52117
3	1.623303	9.056986	0.229358	16.60113	1.705645	-30.5719	1.568699	2.550484
4	-0.08485	1.819444	-0.00987	0.645742	0.192445	3.524902	0.471213	-0.14085
5	0.164361	-5.81165	0.310001	0.424797	0.05151	11.77202	-0.16273	0.065899
6	0.140449	11.36299	0.289083	0.605322	-0.08349	14.43561	-0.01441	0.462136
7	0.07426	0.576836	0.248373	0.141874	0.290386	10.428	0.08389	0.114227
8	-0.00117	-1.25689	0.223732	9.17429	0.115584	36.42863	0.105212	-44.7957
9	-0.15156	3.965171	-0.26817	0.299377	-0.18665	-21.5189	0.052276	15.52571
10	-1.88385	-2.13376	-1.30346	5.931876	-1.51635	-0.97415	0.947068	-36.2199
11	-0.39427	-0.03336	-0.75896	2.010653	-0.75973	-0.62336	-0.36351	-1.42164
12	0.119439	0.865174	-0.15607	0.184525	0.170641	2.878332	-0.13758	-0.03981
13	1.853837	1.817321	1.944267	1.348562	1.264967	17.99155	4.828093	10.3249
14	-1.41939	-2.13148	-1.48265	4.381566	-1.33851	-0.81447	-13.4288	-1.24134

15	0.088622	23.59555	0.005562	0.503597	-0.38955	3.079306	-0.15474	-0.20507
16	-0.53229	0.923846	-0.05075	-0.42229	-0.42198	2.95361	-0.15216	-0.62143
17	-3.37811	5.47062	-3.68037	-4.15058	-2.26637	-3.6243	-5.53142	-5.19723
18	0.250513	0.452106	0.252288	0.203069	0.112478	1.506612	0.212291	-0.18276
19	-0.09756	0.338061	-0.25301	2.280533	-0.56558	0.781255	-0.19516	-0.58901
20	0.160888	4.058534	-0.01326	0.69777	0.204835	3.159134	0.158604	-0.16084
21	0.346371	3.019774	-0.02736	8.808682	1.92197	-33.051	-0.51227	-17.4659
22	0.190165	17.90048	0.499952	0.517902	0.5108	7.97366	0.717233	-19.966
Neurons	16	17	18	19	20	21	22	
1	0.146644	0.467066	0.174628	0.459771	0.438702	1.376009	1.166793	
2	-1.47504	3.073134	-5.98281	-1.47893	-1.17126	7.186292	7.195521	
3	2.1471	-8.13113	-24.8309	-0.16485	1.695354	-8.25545	-9.0933	
4	0.245093	-0.05611	0.046576	0.270803	0.210577	1.549533	8.445993	
5	0.062155	-21.5146	-0.6031	-0.08294	0.128954	1.364465	5.818183	
6	0.217481	1.089691	3.88726	0.297594	0.106225	1.428093	6.537209	
7	0.139645	0.772496	0.280942	0.214426	0.268988	0.338352	3.116008	
8	0.026702	38.04142	-2.73226	0.323631	0.061367	1.896683	0.194209	
9	-0.13336	-6.64631	-28.5791	-0.32191	0.263913	-11.1011	-5.55379	
10	-1.70394	-8.34292	-1.00734	-1.58433	-1.36486	1.238768	1.20343	
11	-0.52364	1.646584	0.257338	-0.83303	-0.74769	6.171906	7.109304	
12	0.306418	0.177796	0.169541	0.254933	0.115372	2.013573	3.42279	
13	1.720825	-1.19602	-61.3717	1.551683	1.823331	-3.62883	-21.8749	
14	-1.70286	-1.67234	-3.35361	-2.3577	-2.86334	-6.72783	-6.34617	
15	-0.04276	1.059667	-1.50711	-0.12116	0.160413	2.313157	2.311381	
16	-0.46909	-0.27087	0.403261	-0.0308	-0.29783	9.485522	3.38731	
17	-4.08283	-14.0739	15.98784	-3.70059	-4.1461	3.726308	0.494004	
18	0.296992	0.459693	0.232083	-0.10371	0.047198	3.222512	3.28792	
19	-0.38269	-0.01095	0.030245	-0.53483	-0.49132	0.584735	0.247924	
20	0.268484	3.389864	1.854776	-0.05905	-0.02748	0.565456	4.23299	
21	1.992298	-28.9998	-20.3103	0.001865	2.19815	-36.1463	15.95618	
22	0.729279	0.699119	6.574418	0.569113	0.175374	1.189134	1.101498	

Link 3

Neurons	Bias	1	2	3	4	5	6	7
1	0.159486	-0.10339	-4.40549	2.494263	0.188992	0.18075	-0.09398	0.302658
2	0.139306	0.11585	-3.45525	11.25747	-0.00856	0.291888	0.063268	-0.09262
3	0.132366	-0.22569	0.387394	-1.54012	0.03479	-0.20204	0.020568	-0.30725
4	-0.20699	-0.05367	-0.90642	0.221163	0.073196	0.261877	0.174607	0.181017

5	-0.19267	-0.27207	-0.00885	-13.8403	0.028598	0.16524	-0.05477	-0.2299
6	-0.13272	0.109165	-3.04315	1.815595	0.220705	0.29141	0.152062	-0.25221
7	0.09637	0.123067	15.37889	-9.37733	0.081858	0.188768	0.427695	-0.31928
8	0.233393	-0.19406	45.28059	4.127724	-0.11309	0.213069	-0.30858	0.188503
9	-0.0673	0.128408	26.91685	5.104981	-0.27377	0.098274	0.055366	0.105874
10	0.011143	0.069976	-10.8406	0.869182	-0.1202	-0.17624	0.134186	-0.07394
11	-0.60225	-0.27494	-19.9487	-14.7642	-0.47789	-0.32021	-0.58652	-0.40559
12	-0.56307	-0.78114	-11.4851	-27.6736	-0.59634	-2.02242	-2.1239	-1.23845
13	-0.548	-0.61086	-35.0262	19.25487	-0.21724	-0.53775	-0.52477	-0.59552
14	0.02356	0.040003	-12.4633	7.700123	-0.05681	0.152126	0.100647	0.050796
15	0.092932	-0.2702	-4.56933	0.809421	0.234512	0.185589	0.225173	0.139636
16	-0.04309	0.278545	11.50678	-2.60028	-0.24203	-0.21942	0.089542	-0.11765
17	0.119949	-0.01422	5.799775	5.328037	0.148029	0.140851	-0.10677	0.115088
18	-0.23303	0.294692	-12.1249	9.40615	0.090459	0.001483	0.053227	0.251074
19	-0.24186	0.147124	1.067953	3.620918	0.256091	-0.23027	-0.14568	0.042714
20	-0.62692	-0.35381	-49.8809	-0.20146	-0.55724	-0.20628	-0.30971	-0.32208
21	-12.7779	-12.3867	-48.6298	50.68315	-1.34533	-1.40491	0.382039	-9.99396
22	-0.10132	0.101312	6.076229	-16.1845	0.227563	-0.23651	0.338249	-0.0456
Neurons	8	9	10	11	12	13	14	15
1	-0.10357	3.078703	0.079717	-0.0539	0.199845	5.727154	-0.73207	-0.25545
2	-0.00343	1.13884	-0.24334	0.158012	0.167367	5.720409	-0.45352	-0.03321
3	0.1757	10.36794	-0.18254	0.023019	0.272429	-4.87488	-14.2637	0.23933
4	0.244618	1.560735	0.128364	0.469327	-0.15787	4.587495	-2.19052	-0.23813
5	-0.29145	-15.175	0.048384	-0.3483	0.062851	-5.39786	-30.4042	-0.21577
6	0.080398	1.039354	0.193822	0.37972	0.265479	4.620126	-0.99679	-0.15528
7	0.083627	6.084118	-0.30462	-0.03715	0.033822	-4.78929	0.285052	-0.25163
8	0.127243	13.25648	0.160016	0.185211	0.286811	-3.56827	0.516533	0.02876
9	0.261059	3.226655	-0.19894	0.468901	0.024255	4.659271	-0.22581	-0.20657
10	0.250816	0.011185	0.120387	0.030908	0.013574	5.791509	-0.84271	0.090622
11	-0.54222	4.262858	-7.47341	-1.11179	-0.40202	-5.97535	8.679685	-0.31848
12	-1.86695	-3.92113	-8.87607	-20.8703	-0.93691	-4.22129	17.08315	-1.11911
13	-0.28099	-0.86447	-0.78057	-0.98184	-0.09708	-5.68015	6.702097	-0.23157
14	0.197181	-16.4143	-0.15542	0.269835	-0.16931	4.800341	-0.40654	0.272488
15	0.237838	-0.2212	-0.07905	-0.02757	-0.12889	5.651057	-1.66659	0.118002
16	0.115929	3.217249	-0.1659	-0.08038	-0.19782	-5.9042	-3.60824	-0.13147
17	0.265833	15.34532	-0.04261	0.097574	0.159149	5.802088	9.290262	0.205064
18	0.870178	-12.9939	0.083011	0.379825	0.280067	4.743563	-0.19606	-0.05933
19	0.500204	-3.2462	0.472105	0.288068	0.097476	4.862567	0.964503	0.311133
20	-0.14595	-0.95627	4.676873	-1.00879	-0.64107	-5.85514	9.086702	-0.2271
21	-14.7962	65.29829	0.443594	-4.19264	-2.36118	-6.58685	3.685859	-11.8304
22	-0.2747	8.475919	-0.08107	-0.20469	-0.30843	-5.73533	2.350766	-0.22652

Neurons	16	17	18	19	20	21	22	
1	0.973056	-0.09946	0.270146	0.238801	0.211436	31.62838	-2.29909	
2	0.140797	-1.03106	0.246294	0.316466	0.079127	30.40626	-2.18871	
3	-0.28004	0.438997	0.156095	0.102684	-0.29388	-34.2645	0.510344	
4	0.146737	-0.01757	-0.09589	-0.13132	-0.18231	-4.68404	0.080443	
5	-0.6684	0.365157	-0.0789	-0.58186	-0.19431	-17.9426	2.463705	
6	0.142308	-0.76443	0.009941	-0.09909	-0.07367	15.17764	-0.4492	
7	-0.40097	0.120193	-0.29058	-0.34193	-0.04404	-25.4237	0.673678	
8	-0.44152	3.683881	0.186688	0.18472	-0.20691	-0.8467	0.005408	
9	0.105025	0.01627	-0.27798	0.008938	0.251122	23.55246	-0.11743	
10	0.184387	0.018311	0.184487	0.047248	0.086534	21.83956	-2.11716	
11	-0.94333	3.841098	-0.21642	-1.29377	-0.31825	-21.3555	1.415344	
12	-0.58215	5.713868	-0.98199	-10.4346	-0.60984	-19.5148	-9.05421	
13	-0.2936	0.479579	-0.5528	-1.21068	-0.21398	26.3757	44.32127	
14	0.557307	-0.31302	-0.24336	-0.02931	0.152001	29.20408	-1.14744	
15	0.741538	-0.11042	0.118086	0.581592	0.304102	23.65648	-2.71327	
16	-0.3974	-0.11186	-0.30228	-0.27639	0.268987	-26.3967	3.017701	
17	0.32176	-0.12397	-0.0703	0.193587	0.227055	30.05043	-2.43997	
18	1.127437	-0.16431	0.134562	0.174262	0.209545	-6.11671	0.003208	
19	0.316728	-0.18947	0.200456	-0.04047	-0.19849	-9.63332	-0.61866	
20	-0.45744	-0.04995	-0.39888	-1.24024	-0.45232	3.727822	1.794568	
21	4.223057	18.27782	0.152369	-5.90944	-1.34664	-66.926	-1.08792	
22	-1.19866	0.660644	0.05915	-0.10588	-0.20276	-26.5335	2.437996	

Link 4

Bias	1	2	3	4	5	6	7	8
-0.07715	-0.244	-0.12092	0.230288	-0.19393	0.250209	-0.24064	0.19609	-1.62205
	9	10	11	12	13	14	15	16
	-0.04737	-0.1826	13.59749	1.142534	-3.43348	-0.06148	-0.30598	0.207477
	17	18	19	20	21	22		
	-0.23099	-0.15542	-0.17053	-8.36234	0.147642	0.263871		

Appendix – II: Data for GA-based-Data-Division Models

Explanation of Categories

T: Training, P: Calibration, V: Validation

Table A2-1: The Data of Network J1 for Screened Data

Am-241	Cs-137	Ash	Category
12345	13854	9.55	T
20186	48606	9.86	T
14847	21690	21.16	T
15462	25490	9.37	T
12513	16159	9.80	T
14385	20186	12.83	T
14705	22195	10.90	T
14525	20543	12.39	T
15135	22743	13.43	T
20151	47894	10.99	T
13743	17992	13.63	T
13344	16895	9.74	T
15289	23738	13.00	T
15540	24991	9.95	T
15416	23875	13.68	T
14553	21171	12.04	T
15230	23505	14.41	T
12763	14432	13.59	T
14255	21073	13.32	T
15171	23441	11.07	T
15981	29029	11.64	T
13163	16393	10.65	T
14273	20120	11.18	T
19965	47322	14.26	T
15751	26726	10.23	T
15235	24015	10.23	T
13973	18639	14.50	T
13984	19305	9.71	T
13430	16627	13.88	T
15358	24145	11.05	T
14692	21166	16.00	T
13514	17105	12.87	T
12732	14413	15.71	P
15929	26954	9.96	P
16727	30487	12.05	P
16475	29566	9.81	P
12058	13163	11.73	P
16282	28447	11.25	P
14042	19242	15.96	P

13154	16276	15.66	V
13744	18118	10.21	V
14310	20176	9.40	V
19042	42837	9.51	V
14550	20932	13.51	V
15171	23321	14.24	V
14628	21238	13.24	V

Figure A2-1. Jump net J1 (tanh) for screened data (GA based)

Table A2-2. Weights of the network

Link 1

Bias	1	2
-0.71224	1.317164	-2.0067

Link 2

bias	1
0.210393	3.104519

Link 3

bias	1	2
-0.00346	1.608049	-0.37752

Explanation of Categories

T: Training, P: Calibration, V: Validation

Table A2-3: The Data of Network S2 for Screened Data

Am	Cs	Ash	Tag
20400	49258	8.90	T
1160	11602	13.75	T
6295	2410	23.33	T
8081	5174	10.33	T
8625	6458	7.48	T
14398	20770	15.48	T
12471	14375	13.91	T
10445	9082	14.59	T
8537	6153	13.78	T
12891	15583	18.03	T
10161	8343	11.89	T
9320	7137	11.66	T
10953	9811	15.43	T
7010	3184	15.23	T
11856	12577	13.61	T
8749	5869	11.70	T
10887	10413	11.10	T
13217	16078	11.33	T
9182	7024	12.32	T
9020	6823	10.88	T
11346	11929	12.44	T
11568	11806	14.21	T
12016	12866	11.66	T
10871	10328	18.26	T
11857	12869	13.96	T
8325	5385	14.60	T
8103	4675	13.28	T
10983	10468	10.29	T
10775	11253	13.47	T
10021	10954	8.74	T
9245	6847	11.58	T
12838	15868	15.36	T
8046	5313	12.20	T
10964	11033	13.89	T
9154	6781	11.49	T
8652	5886	12.85	T
10199	8594	12.93	T
10399	10593	9.75	T
11266	11318	18.04	T
13364	18159	12.31	T
11554	14668	12.67	T

9323	7050	11.73	P
9263	7719	11.60	P
11462	12062	18.81	P
10460	9675	11.73	P
16287	29107	9.40	P
9709	8324	15.42	P
9828	7920	12.18	P
6720	2933	14.12	P
12431	15552	13.65	V
10594	9633	11.52	V
7233	3446	17.75	V
8356	5543	10.94	V
15399	24963	9.75	V
7185	3359	13.53	V
9798	8890	16.79	V
11786	13331	11.26	V

Figure A2-2. The architecture S2 (tanh) for unscreened data (GA)

Table A2-4. Weights for the network

Link 1

bias	1	2
-1.64327	-1.29302	14.29816
18.11653	18.0663	12.41109
-28.7896	-28.6954	-17.4789
19.74525	19.79758	12.29999

Link 2

bias	1	2	3	4
0.057104	-0.31436	1.84755	-0.37503	2.951544
4.207598	4.118149	-0.10122	-0.10248	0.143792
-9.996	-10.09	-0.06698	-0.09791	-0.23129
-10.1694	-10.0789	-0.21386	0.033367	0.05737

Link 3

bias	1	2	3	4
-0.14324	-0.20244	-0.17107	0.340551	0.372982

Explanation of Categories

T: Training, P: Calibration, V: Validation

Table A2-5: The Data of Network J1 for Combined Data (GA)

Am	Cs	Ash	Tag
20400	49258	13.7	T
1160	11602	13.8	T
7010	3184	15.2	T
14847	21690	21.2	T
12431	15552	8.9	T
12058	13163	9.9	T
19965	47322	14.3	T
14525	20543	12.4	T
15171	23441	11.1	T
8046	5313	12.2	T
13430	16627	13.9	T
9828	7920	12.2	T
10460	9675	11.7	T
15462	25490	9.4	T
12471	14375	13.9	T
9154	6781	11.5	T
8524	5985	10.7	T
16287	29107	9.4	T
10594	9633	11.5	T
14398	20770	15.5	T
12891	15583	18	T
15981	29029	11.6	T
11568	11806	14.2	T
9798	8890	16.8	T
8652	5886	12.9	T
13973	18639	14.5	T
14310	20176	9.4	T
8749	5869	11.7	T
15399	24963	9.7	T
11266	11318	18	T
14273	20120	11.2	T
15929	26954	10	T
16475	29566	9.8	T
15289	23738	13	T
13217	16078	11.3	T
14692	21166	16	T
12838	15868	15.4	T
7185	3359	13.5	T
10887	10413	11.1	T
15358	24145	11	T
12016	12866	11.7	T

9020	6823	10.9	T
10983	10468	10.3	T
11346	11929	12.4	T
13514	17105	12.9	T
12513	16159	9.8	T
10161	8343	11.9	T
12732	14413	15.7	T
9320	7137	11.7	T
13163	16393	10.7	T
10964	11033	13.9	T
20151	47894	11	T
10953	9811	15.4	T
11856	12577	13.6	T
8103	4675	13.3	T
12763	14432	13.6	T
10775	11253	13.5	T
15416	23875	13.7	T
9323	7050	11.7	T
11857	12869	14	T
19042	42837	9.5	P
14042	19242	16	P
8325	5385	14.6	P
8081	5174	10.3	P
13743	17992	13.6	P
10445	9082	14.6	P
15230	23505	14.4	P
13154	16276	15.7	P
11786	13331	11.3	P
14553	21171	12	P
15235	24015	10.2	P
14628	21238	13.2	P
7233	3446	17.8	P
8356	5543	10.9	P
15540	24991	10	P
9245	6847	11.6	P
13984	19305	9.7	P
9709	8324	15.4	P
16727	30487	12.1	P
9263	7719	11.6	P
8625	6458	7.5	V
13344	16895	9.7	V
13744	18118	10.2	V
14255	21073	13.3	V
10399	10593	9.8	V
8537	6153	13.8	V
20186	48606	11.7	V
10021	10954	8.7	V

6295	2410	23.3	V
14550	20932	13.5	V
15171	23321	14.2	V
10871	10328	18.3	V
15135	22743	13.4	V
12345	13854	9.6	V
6720	2933	14.1	V
11554	14668	12.7	V
10199	8594	12.9	V
14705	22195	10.9	V
9182	7024	12.3	V
11462	12062	18.8	V
13364	18159	12.3	V
15751	26726	10.2	V
14385	20186	12.8	V
16282	28447	11.3	V

Figure A2-3. Jump net J1 (tanh) for combined data (GA based)

Table A2-6: Weights of J1 network for combined data (GA)

Link 1

Bias	1	2
-2.17859	-0.4668	0.394115
1.982016	0.309649	-0.81825
1.811935	1.34986	14.86431
4.204495	7.095815	0.732328
1.986791	0.309302	-1.03451
2.232216	0.008842	-2.33317
1.254455	-0.70449	-4.23818
3.441808	-13.8118	-12.544
2.055002	0.547276	-0.58585

Link 2

Bias	1	2	3	4	5	6	7	8	9
-0.06878	-0.24144	0.365559	-3.96974	-2.60795	0.073753	-1.0034	-1.05345	-3.57792	0.306871

Link 3

bias	1	2
0.070085	4.704288	-4.3789

Appendix-III: Program Code

Code for crop.cpp

```
//This one cleans the raw text file, such as 36.txt, and converts it
//into a usable file 36gw.txt. This should be run first immediately
after
//files are converted to text.

#include<fstream.h>
#include<stdlib.h> //for exit
void gen_op_file_name(char *ipfile,char *opfile);
void gen_w_op_file_name(char *opfile, char *opwfile);
#define MAX 80
main()
{

char ipfile[20], buffer[MAX],opfile[20],opwfile[20];

cout<<"\nEnter name of file to read: ";
cin>>ipfile;
ifstream raw_data(ipfile);

if (!raw_data)
{
    cout<<"\nError! Couldn't open file.\n";
    exit(-1);
}

// The first four lines are ignored per sampling protocol followed.
// This amounts to 2 seconds of data. The coal was marked
// after 2 seconds of starting the data file.

raw_data.getline(buffer,MAX);
raw_data.getline(buffer,MAX);
raw_data.getline(buffer,MAX);
raw_data.getline(buffer,MAX);

char junk[20];
float intens1,intens2,wt, Wts[40], Int1[40], Int2[40],SumWt=0;

gen_op_file_name(ipfile,opfile);
gen_w_op_file_name(opfile,opwfile);

// generates the name of the op file based on the name of the ip file
// Opfile is 6g.txt for ipfile 6.txt. The 'g' stands for 'Good'

ofstream output(opfile);
ofstream outputW(opwfile);

int i=1;
while(i<=40) //only 40 lines are read
{
```

```

raw_data>>junk>>junk>>wt>>intens1>>intens2>>junk>>junk>>junk>>junk;
    cout<<wt<<"\t"<<intens1<<"\t"<<intens2<<endl;
    output<<wt<<"\t"<<intens1<<"\t"<<intens2<<endl;
    Int1[i-1]=intens1;
    Int2[i-1]=intens2;
    Wts[i-1]=wt;
    SumWt=SumWt+wt;
    i++;
}

// Create Weighted Intensity readings

float SumInt1=0,SumInt2=0;

for (i=0;i<40;i++)
{
    Int1[i]=Int1[i]*Wts[i]/SumWt;
    Int2[i]=Int2[i]*Wts[i]/SumWt;
    SumInt1=SumInt1+Int1[i];
    SumInt2=SumInt2+Int2[i];
    outputW<<Int1[i]<<"\t\t"<<Int2[i]<<endl;
}

outputW<<"\n"<<SumInt1<<"\t\t"<<SumInt2;

cout<<"\nOP file: "<<opfile<<" contains the raw wts and intensity
    readings."<<endl;
cout<<"\nOPW file: "<<opwfile<<" contains weighted intensity
readings."<<endl;

} //end of main

void gen_op_file_name(char *ipfile,char *opfile)
// Adds the letter 'g' to the file name. Therefore, the output
// file for 6.txt becomes 6g.txt

{

int i=0,j=0;
while(j<20)
{
    if (ipfile[i]=='.')
    {
        opfile[j]='g';j++;
    }
    opfile[j]=ipfile[i];
    i++;
    j++;
}
}

```

```

void gen_w_op_file_name(char *opfile, char *opwfile)
{
int i=0,j=0;
while(j<20)
{
    if (opfile[i]=='.')
        {
            opwfile[j]='w';j++;
        }
    opwfile[j]=opfile[i];
    i++;
    j++;
}

} //end of function

```

----- End of crop.cpp -----

Code for gen_data_file.cpp

```
//crop.cpp should be run before this file is run. crop.cpp generates
// the *.gw files that are input here.
//This file creates the final ANN input data file required for training
//The output for training is created from the Excel spreadsheet

#include<fstream.h>
#include<stdlib.h>
void obtain_file_name(char *file_name, int i);
main()
{

    int i=0,k=0,start=6,end,SIZE;
    char fn[10], junk[20];
    float r1,r2;

    cout<<"\nFile numbers start with 6.txt.  Enter the number in the
    ending file numbers: ";
    cin>>end;
    cout<<endl<<endl;

    if (end>=14)
        SIZE=end-start+1;
    else
        SIZE=end-start;
    float readings[SIZE][2];
    k=0;
    for (i=start;i<=end;i++)
    {
        if ((i==14)|| (i==30) || (i==37)) continue;  //skip files;
        obtain_file_name(fn,i);
        cout<<"\ni: "<<i<<"\tFilename: "<<fn<<"\t";
        ifstream data(fn);
        ofstream out("output.dat",ios::app);
        if (!data)
        {
            cout<<"\nError!Couldn't open input file.\n";
            exit(-1);
        }

        if (!out)
        {
            cout<<"\nError!Couldn't open output file.\n";
            exit(-1);
        }

        for (int j=1;j<=80;j++)
            data>>junk; //ignoring the top 40 lines

        data>>readings[k][0]>>readings[k][1]; //reading 81st and 82nd fields
        out<<readings[k][0]<<"\t"<<readings[k][1]<<endl;
        k++;
    }
}
```

```

}

} //end of main
////////////////////////////////////
void obtain_file_name(char *file_name, int i)
{
    if (i==6) strcpy(file_name, "6gw.txt");
    else if (i==7) strcpy(file_name, "7gw.txt");
    else if (i==8) strcpy(file_name, "8gw.txt");
    else if (i==9) strcpy(file_name, "9gw.txt");
    else if (i==10) strcpy(file_name, "10gw.txt");
    else if (i==11) strcpy(file_name, "11gw.txt");
    else if (i==12) strcpy(file_name, "12gw.txt");
    else if (i==13) strcpy(file_name, "13gw.txt");
    else if (i==15) strcpy(file_name, "15gw.txt"); // #14 doesn't exist
    else if (i==16) strcpy(file_name, "16gw.txt");
    else if (i==17) strcpy(file_name, "17gw.txt");
    else if (i==18) strcpy(file_name, "18gw.txt");
    else if (i==19) strcpy(file_name, "19gw.txt");
    else if (i==20) strcpy(file_name, "20gw.txt");
    else if (i==21) strcpy(file_name, "21gw.txt");
    else if (i==22) strcpy(file_name, "22gw.txt");
    else if (i==23) strcpy(file_name, "23gw.txt");
    else if (i==24) strcpy(file_name, "24gw.txt");
    else if (i==25) strcpy(file_name, "25gw.txt");
    else if (i==26) strcpy(file_name, "26gw.txt");
    else if (i==27) strcpy(file_name, "27gw.txt");
    else if (i==28) strcpy(file_name, "28gw.txt");
    else if (i==29) strcpy(file_name, "29gw.txt"); // #30 doesn't exist
    else if (i==31) strcpy(file_name, "31gw.txt");
    else if (i==32) strcpy(file_name, "32gw.txt");
    else if (i==33) strcpy(file_name, "33gw.txt");
    else if (i==34) strcpy(file_name, "34gw.txt");
    else if (i==35) strcpy(file_name, "35gw.txt");
    else if (i==36) strcpy(file_name, "36gw.txt");
    else if (i==38) strcpy(file_name, "38gw.txt"); // #37 doesn't exist
    else if (i==39) strcpy(file_name, "39gw.txt");
    else if (i==40) strcpy(file_name, "40gw.txt");
    else if (i==41) strcpy(file_name, "41gw.txt");
    else if (i==42) strcpy(file_name, "42gw.txt");
    else if (i==43) strcpy(file_name, "43gw.txt");
    else if (i==44) strcpy(file_name, "44gw.txt");
    else if (i==45) strcpy(file_name, "45gw.txt");
    else if (i==46) strcpy(file_name, "46gw.txt");
    else if (i==47) strcpy(file_name, "47gw.txt");
    else if (i==48) strcpy(file_name, "48gw.txt");
    else if (i==49) strcpy(file_name, "49gw.txt");
    else if (i==50) strcpy(file_name, "50gw.txt");
    else if (i==51) strcpy(file_name, "51gw.txt");
    else if (i==52) strcpy(file_name, "52gw.txt");
    else if (i==53) strcpy(file_name, "53gw.txt");
    else if (i==54) strcpy(file_name, "54gw.txt");
    else if (i==55) strcpy(file_name, "55gw.txt");
    else if (i==56) strcpy(file_name, "56gw.txt");
    else if (i==57) strcpy(file_name, "57gw.txt");
    else if (i==58) strcpy(file_name, "58gw.txt");

```

```

else if (i==59) strcpy(file_name,"59gw.txt");
else if (i==60) strcpy(file_name,"60gw.txt");
else if (i==61) strcpy(file_name,"61gw.txt");
else if (i==62) strcpy(file_name,"62gw.txt");
else if (i==63) strcpy(file_name,"63gw.txt");
else if (i==64) strcpy(file_name,"64gw.txt");
else if (i==65) strcpy(file_name,"65gw.txt");
else if (i==66) strcpy(file_name,"66gw.txt");
else if (i==67) strcpy(file_name,"67gw.txt");
else if (i==68) strcpy(file_name,"68gw.txt");
else if (i==69) strcpy(file_name,"69gw.txt");
else if (i==70) strcpy(file_name,"70gw.txt");
else if (i==71) strcpy(file_name,"71gw.txt");
else if (i==72) strcpy(file_name,"72gw.txt");
else if (i==73) strcpy(file_name,"73gw.txt");
else if (i==74) strcpy(file_name,"74gw.txt");
else if (i==75) strcpy(file_name,"75gw.txt");
else if (i==76) strcpy(file_name,"76gw.txt");
else if (i==77) strcpy(file_name,"77gw.txt");
else if (i==78) strcpy(file_name,"78gw.txt");
else if (i==79) strcpy(file_name,"79gw.txt");
else if (i==80) strcpy(file_name,"80gw.txt");
else if (i==81) strcpy(file_name,"81gw.txt");
else if (i==82) strcpy(file_name,"82gw.txt");
else if (i==83) strcpy(file_name,"83gw.txt");
else if (i==84) strcpy(file_name,"84gw.txt");
else if (i==85) strcpy(file_name,"85gw.txt");
else if (i==86) strcpy(file_name,"86gw.txt");
else if (i==87) strcpy(file_name,"87gw.txt");
else if (i==88) strcpy(file_name,"88gw.txt");
else if (i==89) strcpy(file_name,"89gw.txt");
else if (i==90) strcpy(file_name,"90gw.txt");
else if (i==91) strcpy(file_name,"91gw.txt");
else if (i==92) strcpy(file_name,"92gw.txt");
else if (i==93) strcpy(file_name,"93gw.txt");
else if (i==94) strcpy(file_name,"94gw.txt");
else if (i==95) strcpy(file_name,"95gw.txt");
else if (i==96) strcpy(file_name,"96gw.txt");
else if (i==97) strcpy(file_name,"97gw.txt");
else if (i==98) strcpy(file_name,"98gw.txt");
else if (i==99) strcpy(file_name,"99gw.txt");

}

```

The Recommended Neural Network

Code for ucm.cpp

Note: Requires Combo_GA.h file (the fitted combined data model)

```
#include<fstream.h>
#include<stdio.h> //for fseek()
#include<stdlib.h> //for exit
#include<string.h> //for strlen()
#include<math.h> //for ceil()
#include "Combo_GA.h"

#define MAX 80
#define MIN_WIDTH 35 // min. line width for it to count
#define MAX_NULL 5 // max number of NULL lines allowed
main()
{
// !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
// !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

char *ipfile="test.txt";
char *opfile="ash.txt";
char junk[MAX] = "abcde";
int j=0,len, composite = 0,reset=1, coal_type=1,Empty_Choice;
double intens1=0,intens2=0,wt=0, Am_sum=0, Cs_sum=0, Wt_sum=0;
double input[2],true_ash, Ave_Ash=0, Empty_Am=0, Empty_Cs=0;
streampos location;

// !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
// !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

cout <<
"\n*****
\nThis software was developed and written by Dr. Rajive Ganguli,\n
University of Alaska Fairbanks under the Department of Energy \n
grant DE-FC26-01NT41058. Use this software at your own risk.
\n*****
\n\n";

cout<<"\nEnter the name of the analyzer file to read: ";
cin>>ipfile;

ifstream raw_data(ipfile); //opening the file in text mode
if (!raw_data)
{
cout<<"\nError! Couldn't open file.\n";
exit(-1);
}

cout<<"\nEnter the name of the file to write ash data to: ";
cin>>opfile;
ofstream ash(opfile);
```



```

//cout<<"\nEnter coal type.\nEnter 1 for Unscreened, 2 for screened and
0 if you dont know: ";
//cin>>coal_type;

cout<<"\nMake empty belt adjustments (i.e. ignore readings that \nare
equal to empty belt readings)?";
cout<<"\n1 for yes and 2 for no\n";
cin>>Empty_Choice;

if (Empty_Choice == 1)
{
    cout<<"\nEnter Am and Cs counts for empty belt.\n";
    cin>>Empty_Am>>Empty_Cs;
}

// Initially, we check to see if the present line has a minimum width
// To check for that, the line has to be read, which advances the
// file pointer to the end of the line. Therefore, after the line is
// read and its width determined, if it is wide enough, we push the
file
// pointer back to where it was before it read the line, to enable it
to
// re-read the line, this time,intelligently and into various fields.

j=0;
while(reset < MAX_NULL)      {
    location = raw_data.tellg();
    raw_data.getline(junk,MAX,'\n');
    len = strlen(junk);

    if (len==0)
    {
        reset++;
    }
    else if ( len >= MIN_WIDTH )
    {
        reset = 0;
        raw_data.seekg(location); //go back to previous location

raw_data>>junk>>junk>>wt>>intens1>>intens2>>junk>>junk>>junk>>junk;

        // checking to see if the read fields are valid!!
        // if the intensities are too low, obviously there is a
problem

        if ( (intens1>900) && (intens2>900) )
        {
            ///////////////////////////////////
            // Compositing 40 Am and Cs values to obtain one Am
and Cs value
            // This modifies the data into 20 sec samples, like
the ones
            // used to develop the neural network
            ///////////////////////////////////

```

```

// Disregard empty belt readings.  If Empty_Choice
==2, then the IF
// statement will always be FALSE as intens1 and
intens2 will not be < 0
if (      (fabs(intens1 - Empty_Am) < 0.01*Empty_Am)
&&
      (fabs(intens2 - Empty_Cs) < 0.01*Empty_Cs)  )
    continue;

Am_sum = Am_sum + wt*intens1;
Cs_sum = Cs_sum + wt*intens2;
Wt_sum = Wt_sum + wt;

composite++;
if (composite ==40)
{
    input[0] =  ceil (Am_sum / Wt_sum);
    input[1] =  ceil (Cs_sum / Wt_sum);
    ash<<input[0]<<"\t"<<input[1]<<"\t";  //input
gets modified in function

    Combo_GA(input, &true_ash); //using NN for both
screen and unscreened

    Ave_Ash = Ave_Ash + true_ash;
    ash<<true_ash<<endl;
    composite = 0;
    Wt_sum = 0; Am_sum=0;Cs_sum=0;
    j++;
}
} //end of IF ensuring the right fields are read

} //end of IF checking line width

} //end of while

raw_data.close();

cout<<"\nThe average ash is: "<<Ave_Ash/j;
cout<<"\nThe number of valid ash computations (each equiv to 20 secs)
:"<<j<<endl;

} //end of main

////////////////////////////////////
////////////////////////////////////
////////////////////////////////////
////////////////////////////////////

```

CODE for Combo_GA.h (primarily from Neurshell 2)

```
void Combo_GA(double *inarray, double *outarray)
{
    double netsum;
    double feature2[12];
    double feature3[12];

    /* inarray[0] is Am */
    /* inarray[1] is Cs */
    /* outarray[0] is Ash */

    if (inarray[0]<1160) inarray[0] = 1160;
    if (inarray[0]>20400) inarray[0] = 20400;
    inarray[0] = 2 * (inarray[0] - 1160) / 19240 -1;

    if (inarray[1]<2410) inarray[1] = 2410;
    if (inarray[1]>49258) inarray[1] = 49258;
    inarray[1] = 2 * (inarray[1] - 2410) / 46848 -1;

    netsum = -7.317823E-02;
    netsum += inarray[0] * 0.352372;
    netsum += inarray[1] * -0.2556932;
    feature2[0] = exp(-netsum * netsum);

    netsum = -8.153406E-03;
    netsum += inarray[0] * -3.818367E-02;
    netsum += inarray[1] * -6.934065E-02;
    feature2[1] = exp(-netsum * netsum);

    netsum = -0.1862512;
    netsum += inarray[0] * -0.1198367;
    netsum += inarray[1] * 1.649883E-02;
    feature2[2] = exp(-netsum * netsum);

    netsum = 15.58185;
    netsum += inarray[0] * 15.1056;
    netsum += inarray[1] * 22.08637;
    feature2[3] = exp(-netsum * netsum);

    netsum = -0.529511;
    netsum += inarray[0] * -0.8256759;
    netsum += inarray[1] * -3.358302;
    feature2[4] = exp(-netsum * netsum);

    netsum = -5.328222;
    netsum += inarray[0] * -5.397474;
    netsum += inarray[1] * -2.85527;
    feature2[5] = exp(-netsum * netsum);

    netsum = -0.4790896;
    netsum += inarray[0] * -0.6008766;
    netsum += inarray[1] * 0.1528852;
    feature2[6] = exp(-netsum * netsum);

    netsum = -63.38087;
    netsum += inarray[0] * -63.81889;
    netsum += inarray[1] * -68.52663;
```

```

feature2[7] = exp(-netsum * netsum);
netsum = -25.73829;
netsum += inarray[0] * -25.63888;
netsum += inarray[1] * -8.241678;
feature2[8] = exp(-netsum * netsum);

netsum = -21.56628;
netsum += inarray[0] * -21.74683;
netsum += inarray[1] * -36.46988;
feature2[9] = exp(-netsum * netsum);

netsum = -0.3153108;
netsum += inarray[0] * -0.3483258;
netsum += inarray[1] * 0.9670362;
feature2[10] = exp(-netsum * netsum);

netsum = 38.22894;
netsum += inarray[0] * 37.71638;
netsum += inarray[1] * 0.8436445;
feature2[11] = exp(-netsum * netsum);

netsum = -0.3994672;
netsum += inarray[0] * -0.4696536;
netsum += inarray[1] * 0.0956169;
feature3[0] = 1 - exp(-netsum * netsum);

netsum = -0.012382;
netsum += inarray[0] * -7.385242E-02;
netsum += inarray[1] * -0.1314417;
feature3[1] = 1 - exp(-netsum * netsum);

netsum = 34.34132;
netsum += inarray[0] * 33.86886;
netsum += inarray[1] * 0.7776579;
feature3[2] = 1 - exp(-netsum * netsum);

netsum = 0.4279565;
netsum += inarray[0] * -0.1648474;
netsum += inarray[1] * 0.0954555;
feature3[3] = 1 - exp(-netsum * netsum);

netsum = 0.192949;
netsum += inarray[0] * 0.1444058;
netsum += inarray[1] * 3.698409E-02;
feature3[4] = 1 - exp(-netsum * netsum);

netsum = -0.3689938;
netsum += inarray[0] * -0.5703608;
netsum += inarray[1] * 0.1493886;
feature3[5] = 1 - exp(-netsum * netsum);

netsum = -83.31644;
netsum += inarray[0] * -83.38815;
netsum += inarray[1] * -90.50157;
feature3[6] = 1 - exp(-netsum * netsum);

netsum = -8.003247E-02;

```

```

netsum += inarray[0] * 0.1129849;
netsum += inarray[1] * 0.2769367;
feature3[7] = 1 - exp(-netsum * netsum);

netsum = 0.3729149;
netsum += inarray[0] * 0.609257;
netsum += inarray[1] * -0.136893;
feature3[8] = 1 - exp(-netsum * netsum);

netsum = 23.36234;
netsum += inarray[0] * 23.36082;
netsum += inarray[1] * 6.973737;
feature3[9] = 1 - exp(-netsum * netsum);

netsum = -34.38374;
netsum += inarray[0] * -33.96209;
netsum += inarray[1] * -0.6939385;
feature3[10] = 1 - exp(-netsum * netsum);

netsum = -0.1086157;
netsum += inarray[0] * -4.317177E-02;
netsum += inarray[1] * 0.183918;
feature3[11] = 1 - exp(-netsum * netsum);

netsum = 0.1684675;
netsum += feature2[0] * 0.1800401;
netsum += feature2[1] * 0.0102162;
netsum += feature2[2] * -0.1203788;
netsum += feature2[3] * 1.985458E-02;
netsum += feature2[4] * 0.2116783;
netsum += feature2[5] * 0.4327381;
netsum += feature2[6] * 0.2527907;
netsum += feature2[7] * -1.058036;
netsum += feature2[8] * -0.8932153;
netsum += feature2[9] * 1.150584;
netsum += feature2[10] * -2.019392;
netsum += feature2[11] * 1.727629;
netsum += -0.3650914;
netsum += feature3[0] * -0.490962;
netsum += feature3[1] * -0.2484267;
netsum += feature3[2] * -5.040262E-02;
netsum += feature3[3] * -2.903791;
netsum += feature3[4] * -2.733486;
netsum += feature3[5] * 2.713592;
netsum += feature3[6] * -0.289025;
netsum += feature3[7] * 3.506965;
netsum += feature3[8] * 1.469323;
netsum += feature3[9] * 9.306035E-02;
netsum += feature3[10] * -0.0103562;
netsum += feature3[11] * -0.2611186;
outarray[0] = 1 / (1 + exp(-netsum));

outarray[0] = 15.8 * (outarray[0] - .1) / .8 + 7.5;
if (outarray[0]<7.5) outarray[0] = 7.5;
if (outarray[0]>23.3) outarray[0] = 23.3;
} //end of function

```