MULTIPLE ALGORITHM SIGNAL DETECTION USING NEURAL NETWORKS

Bion J. Merchant, [1] Jeffery T. Drake, [2] Darren Hart, [1] and Christopher J. Young[1]

Sandia National Laboratories[1], NASA Fellow, New Mexico State University[2]

Sponsored by National Nuclear Security Administration
Office of Nonproliferation Research and Engineering
Office of Defense Nuclear Nonproliferation

Contract No. DE-AC04-94AL85000

ABSTRACT

Nuclear explosion and non-proliferation monitoring require the processing of huge amounts of seismic sensor data, and the amount of data is increasing due to the continual, global expansion of monitoring stations. This overwhelming amount of data presents a great challenge to monitoring organizations that must detect and characterize seismic events. Given the limited number of analysts, detection is generally performed using automated techniques, freeing analysts to focus on other, more complex, problems. In this paper we evaluate the automated detection performance of a wide variety of algorithms and present results of fusing the best of these algorithms with a supervised neural network, forming a hybrid detector designed to take advantage of the "best" attributes of many algorithms.

Many different algorithms have been proposed for use in detecting seismic events. However, each algorithm has its own set of strengths and weaknesses. There is no single detection method that is guaranteed to work in all cases. Some researchers have begun simultaneously applying several different algorithms that complement one another. This is the approach that we take in this paper. We evaluate multiple detection algorithms to determine which are best suited for use together in automated detection.

We evaluate and present detection performance, using several detection algorithms as inputs, to a supervised neural network "trained" to predict the detection of an event. A neural network consists of multiple levels of input weights and non-linear functions that can be optimized to minimize the error between the output of the network and some ground-truth training data. Such neural networks can be designed and "trained" to detect events for a geographic region and optimized for any geological peculiarities of the region. The performance of the neural network is then compared to the results obtained by an analyst reviewing the same data. The data we use consists of events recorded by the New Mexico Institute of Mining and Technology network. These events include mining explosions from mines in New Mexico and eastern Arizona as well as earthquake activity throughout the region.

OBJECTIVE

The objective of this study is to investigate the use of neural networks for detecting seismic events. Our eventual intent is to develop neural network detectors for use in an automated processing pipeline for nuclear explosion and nonproliferation monitoring. In this paper, we will be investigating the process involved in preparing and training neural networks for use in seismic event detection.

Currently most automated processing pipelines detect seismic events through the use of a single algorithm that is computed from the incoming seismic waveform. One such commonly used algorithm is the STA/LTA. In order to detect an event, the real-time process looks for some change in the detection algorithm. This change can be defined in several ways, such as the slope of the algorithm output or the algorithms current level. The triggering method requires that an arbitrary threshold level be defined. If the threshold is too high, then events will go undetected and if the threshold is too low, the system will be overwhelmed by false-alarms. The optimal threshold level depends not only on the algorithm being used but also the seismicity of the particular geographic region that is being monitored.

A neural network can be trained to identify seismic events by distinguishing between the algorithm output from signal and noise waveforms. Rather than simply looking at the level of the detection algorithm, a neural network is trained to look for particular patterns in the algorithm. In the training process, a newly initialized neural network is presented with input patterns from both signal and the noise segments from the detection algorithms. The network parameters can then be optimized to best predict the source type for each of those input patterns.

We will be evaluating neural network detection using seismic data recorded by the New Mexico Tech Seismic Network (NMTSN). The data we will be using consists of mining explosions from a mine near Santa Rita in southern New Mexico during August 1997. There are 29 seismic events available from that time period. We will use a subset of those events to train a neural network and then run the network on the remaining events to evaluate the performance of the neural network.

Neural Networks

The type of neural network that we will be making use of in this study is called a feedforward, multi-layer network. The network consists of an arbitrary number of inputs nodes, a single output node, and one or more hidden layers. Each of the hidden layers contains a pre-defined number of nodes. Weighted links connect each node within a layer to all of the nodes within the next layer. The neural network also contains transfer functions between the layers to scale the outputs of each layer to a pre-defined range. The transfer function that we will be using is called a tansig function, which is similar to the hyperbolic tangent. This transfer function has the effect of scaling the outputs from -1 to 1, which simplifies the process of interpreting the output of the network.

The weighted links between the nodes can be adjusted to produce some desired output for a given set of inputs. This adjustment procedure is called the training of the neural network. So, a neural network can be thought of as a functional mapping between a domain of inputs and a range of outputs. In our case, the set of inputs consist of the outputs from several algorithms. We take a short window of the algorithm output for both a section of noisy waveform data and the initial onset of the seismic event signal. The noise inputs are mapped to a zero on the output of the neural network and the signal inputs are mapped to a one. The neural network is then trained to minimize the mean-squared error between its predicted outputs and our defined training outputs. This choice of a zero output for noise and a one output for signal allows for the neural network output to be interpreted as a probability of an event detection. The output of the neural network still requires some threshold level to trigger a detection, however selecting an appropriate probability level is simpler than selecting such a level on an STA/LTA where the values can range from 0 to infinite.

Determining the optimal size, the number of layers and the number of nodes in each layer, of a neural network is a difficult problem. There are many heuristics that have been suggested that relate the amount of training data to the number of weighted links or relate the number of inputs to the number of nodes in each layer. A suitable solution is to simply start with a small network and gradually increase the size until there is no further improvement in performance. However, this is generally a lengthy process involving trial and error.

Once a neural network has been initialized and trained, data can be run through the network a window length at a time. So, at each time step, a window of data will be loaded into the inputs of the neural network. The neural network will then output a single detection probability value. The process will then move over one time step and load the next window of data into the network. This continues indefinitely with the neural network producing a series of detection probabilities. For a given set of input algorithms, a neural network will generate an output sequence in which a value of one indicates the presence of an event and a value of zero indicates the absence of an event.

RESEARCH ACCOMPLISHED

The data that we will be using from the NMTSN is sampled at 100 Hz, however we have found that the usable frequency content is typically much less. So, the data is down-sampled to 25 Hz to reduce the amount of data that needs to be processed. In addition, the data is bandpass filtered from 2 to 8 Hz with a third-order Butterworth filter. For training the neural network, we are using a 0.5 second (13 sample) window of data centered on the Pg arrival for the signal window and starting 5 seconds in advance of the Pg arrival for the noise window. The Pg arrival is not necessarily the first arrival for these events, however for this dataset it has a much higher energy content than either the P or the Pn arrivals. So, we chose to window around this arrival because the algorithms appear to exhibit their desired responses during the Pg arrival. A different set of data might have a more appropriate arrival to use. By selecting the noise window 5 seconds in advance of the Pg, we ensure that the noise used in training the network is free from any portion of the signal.

Since the window length of the data is 13 samples long, any neural network that we construct will have 13 input nodes for each algorithm that is to be loaded into the network. If we were using two algorithms, the network would have 26 inputs. If there were 3 algorithms in use, the network would have 39 inputs, and so on.

The 29 events that we are using have been divided into 4 groups: training, testing, validation, and running. The training group contains 14 events and is used in the training process to evaluate how to adjust the network link weights so as to minimize the prediction error. The testing and validation groups each contain 5 events and are used during the training process to ensure that the network does not become overly trained for the set of training data. Finally, the running group contains the remaining 5 events, which are run concurrently through the neural network to determine the output sequence of the network for those events. This division of the data into groups allows us to evaluate the performance of the neural network independently of the data on which the network is being trained. Otherwise, any network that is created runs the risk of being overly trained for a specific set of data.

Below, in Figure 1, is a typical plot of the performance of a neural network as it is being trained. After several training epochs (an epoch being a single iteration in the optimization process) the performance of the training and validation datasets begins to level off while the training dataset is demonstrating further improvement. This would tend to indicate that the neural network is beginning to become overly optimized for the particular training set. At this point, further training provides little benefit.

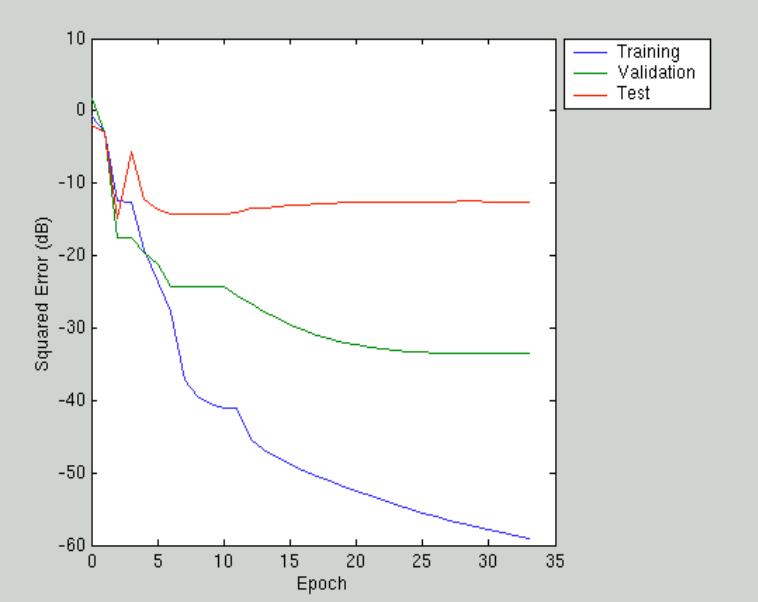

Figure 1. Typical performance of a neural network over time as it is being trained.

In order to form a neural network, there are several steps that must first be taken in preparation: select the algorithms to use as inputs to the neural network, piece out the section of data to use for training, and appropriately scale the inputs to normalize out differences in magnitude between events.

Selection of algorithms

We first had to select the algorithms that we wanted to use as inputs to the neural network. We have initially arbitrarily chosen several algorithms that exhibit significant changes in the presence of signal versus noise. The algorithms that we chose to examine for this study were the change in autoregressive coefficients (ARC) (Wang, 1997), kurtosis (Savvaidis, 2002), and STA/LTA.

Autoregressive coefficients are calculated by modeling the waveform as an autoregressive process. We chose to make use of a fourth-order AR model. At each time-step the AR coefficients are determined from the four previous samples. The ARC is then calculated to be the normalized difference between this time-steps AR coefficients and the previous time-steps AR coefficients. So, the ARC indicates how much the signal model changes between time-steps.

$$ARC = \frac{\left| \sum_{k=1}^{4} w_k(i) - \sum_{k=1}^{4} w_k(i-1) \right|}{\left| \sum_{k=1}^{4} w_k(i) \right|},$$

where $w_k(i)$ for k = 1..4 are the fourth-order AR coefficients at time i.

Kurtosis is measured as the ratio between the fourth-order central moment and the second-order central moment. It is typically interpreted as a measure of how gaussian a signal is. Kurtosis has a value of 0 for a gaussian signal and it increases as the signal becomes increasingly non-gaussian. This algorithm is included to take advantage of the tendency for noise to be gaussian and seismic events to be strongly non-gaussian.

An STA/LTA is the ratio between the short-term average energy and the long-term average energy and is the most commonly used method of detecting seismic events. Spikes within an STA/LTA indicate an abrupt increase in the

level of short-term energy. We make use of a 10 second long-term window and a 1 second short-term window. Examples of these three algorithm outputs are shown in Figure 2.

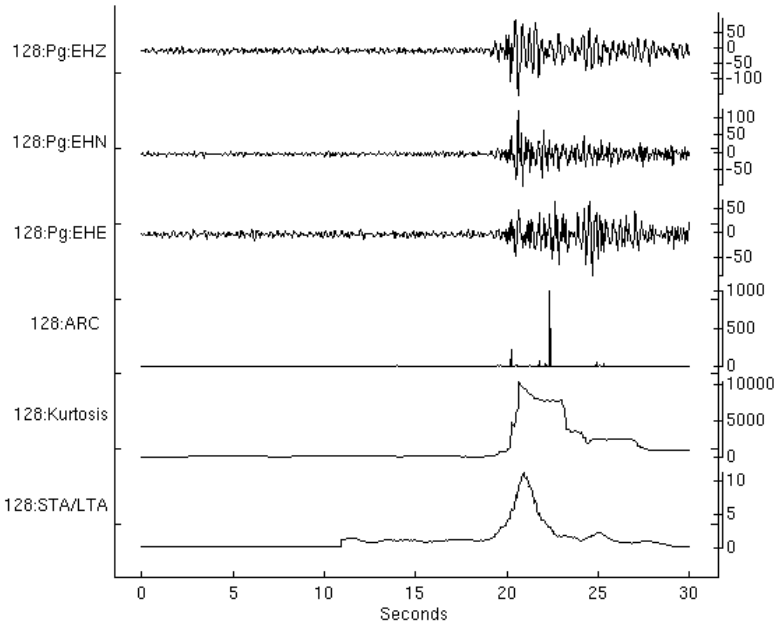

Figure 2. Waveforms and algorithms for one of the events. Three component waveform data is show, however only the z-component is used in computing the algorithms.

Time alignment of training data

A crucial step in the training of a neural network is selecting a portion of the waveform as training data and designating that window as being either signal or noise. If the training windows are poorly chosen, then they might not capture the feature of interest within the detection algorithm. This would inhibit the neural network from being able to properly identify the key features of the algorithm that indicate the presence of a signal. An example of a misaligned training window can be seen in Figure 3. One of the signal patterns was not lined up properly and so appears to be similar to the noise patterns.

To help facilitate the training window selection, we needed to have very precise arrival picks to identify the signal onset. To assist in the selection of the training windows, these events were hand picked and relocated by a seismic analyst familiar with the NMTSN. In addition, the training process requires some operator-assisted evaluation to verify that the training sets are consistent.

Neural network input scaling

A second key issue when preparing data for a neural network is how to scale the input patterns prior to loading them into the network. This issue applies to both training data and to data that is being run through the neural network. For many applications using neural networks, directly scaling the set of inputs is strongly discouraged. This makes sense when the inputs have some physical meaning such as distance or temperature. However, for the algorithms that we are using, the particular values of the detection algorithms have no meaning in and of themselves. Instead, the values are only meaningful in relationship to the values of the adjacent algorithm time-samples. The intent in applying some type of scaling to each window of data that is loaded into the network is to normalize for any differences in the relative magnitudes of the algorithms. This should help to make the neural networks detection capability invariant to differences in signal-to-noise ratios (SNR) between events.

For example, if we were examining an STA/LTA for two events, one of which has an SNR of 10 and the other has an SNR of 100, the smaller event would appear nearly flat relative to the much larger event. However, we desire to be able to detect either event equally well, regardless of the difference in SNR between them. So, we have chosen to scale the neural network inputs to eliminate variations in the level of the algorithms. This allows us to focus solely on relative changes in the algorithms.

Figure 3. Overlayed Kurtosis training patterns for signal (left) and noise (right) for the 14 training events.

In order to determine the method of scaling that achieves the best results, we built neural networks using several different methods of scaling: no scaling, normalize by the maximum, normalize to between zero and 1, and normalize to a mean of zero and a standard deviation of one. For each algorithm we will be presenting the scaling method that achieved that best results.

What we found was that, in general, the kurtosis and STA/LTA algorithms performed best when each window of data to be fed into the neural network was normalized by its maximum value. This normalization resulted in the neural networks being trained to detect transitions within the algorithm regardless of the particular value of the algorithm. The ARC algorithm had the best output when no scaling was applied to the input algorithm. However, as will be seen later, the ARC algorithm did not perform as well as kurtosis and STA/LTA.

Single algorithm neural networks

Initially, we began investigating the use of neural networks by evaluating the performance of neural networks that employ only a single input algorithm. We will be able to determine the optimal parameters for each algorithm individually as well as providing a basis for comparing the results from multiple algorithm neural networks. For each of the test runs, we formed a fairly simple neural network with 2 layers and 13 nodes within each layer. Determining the size of the neural network was a fairly arbitrary process. We ran simulations using different numbers of layers and nodes per layer and typically found only minor variations between the outputs of the networks. The size we chose was one that appeared to work well for all of the algorithms that we were using.

However, it is important to stress that there is no single optimal network configuration. The inputs algorithms and neural network outputs for each of the 5 events that were run through the network are shown in Figures 4-7.

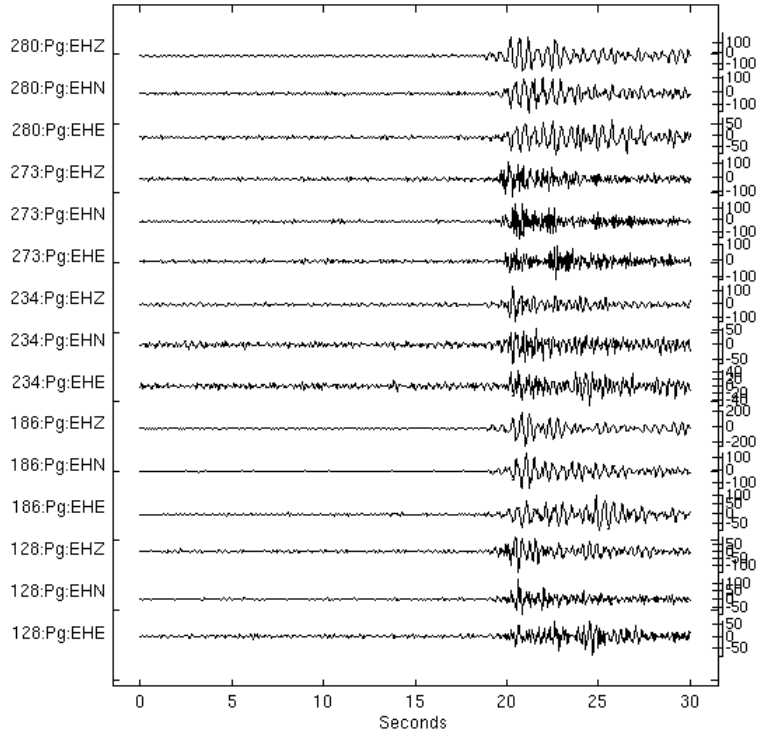

Figure 4. Seismic waveform data for the 5 events to be run in a neural network.

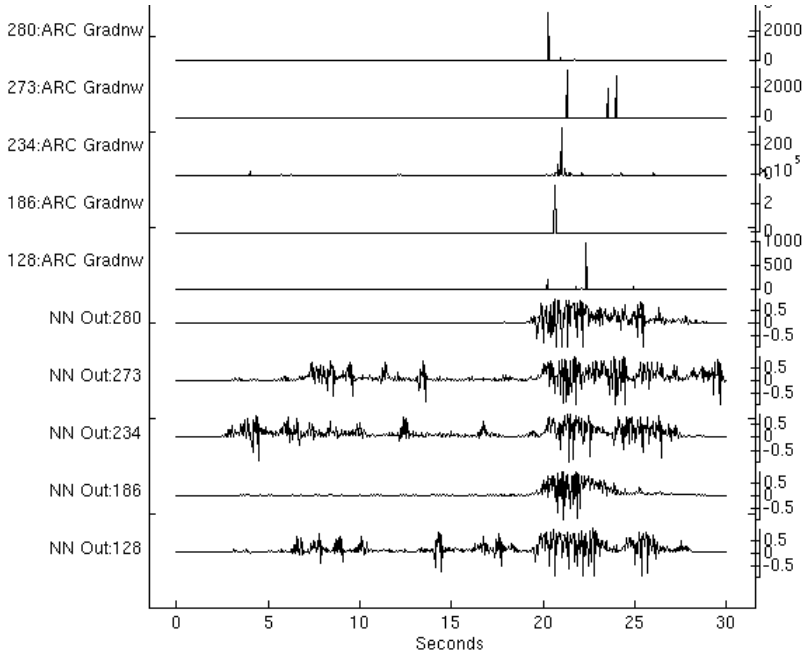

Figure 5. ARC processed data and neural network outputs for the data shown in Figure 4.

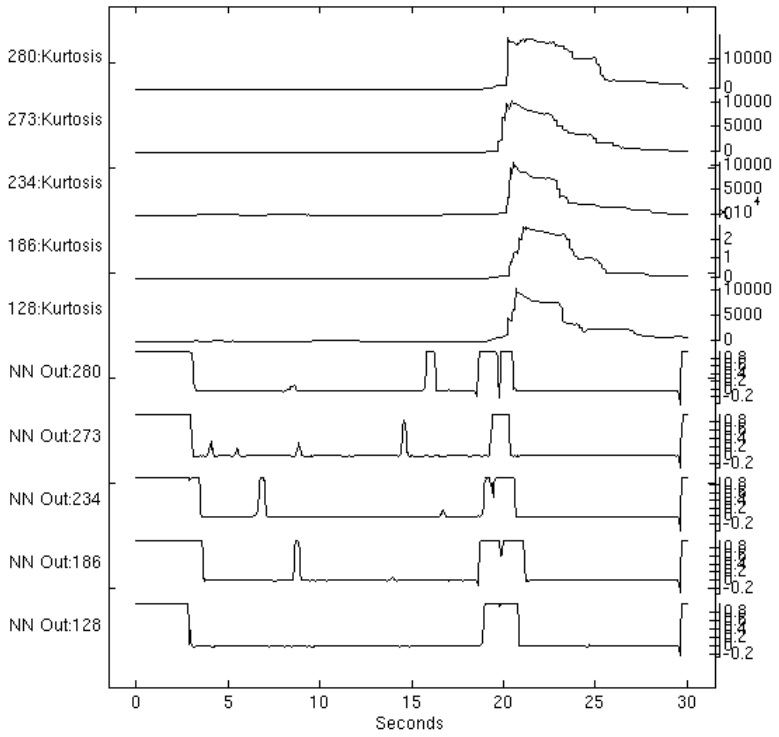

Figure 6. Kurtosis processed data and neural network outputs for the data shown in Figure 4.

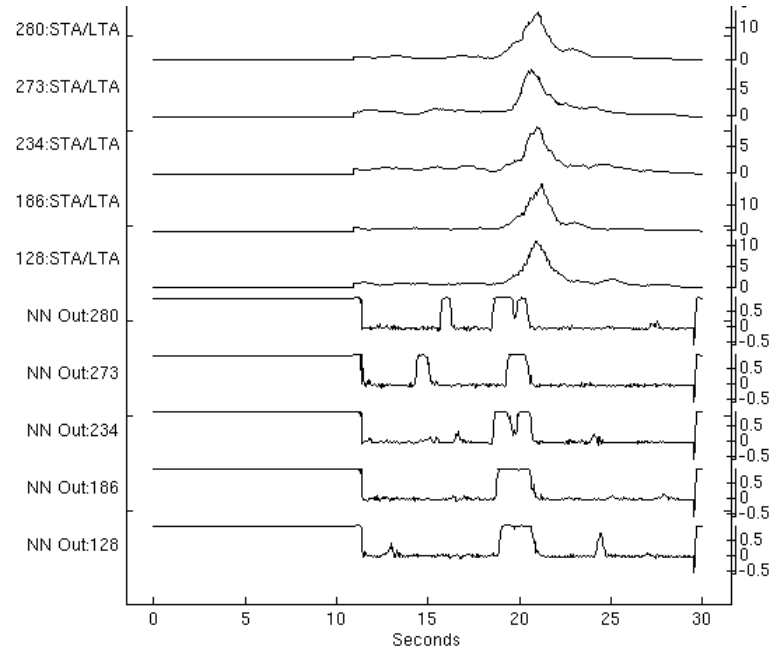

Figure 7. STA/LTA processed data and neural network outputs for the data shown in Figure 4.

Ideally, what we would expect to see in Figures 5, 6, and 7 above, would be for the neural network output to be zero prior to the arrival, one during the arrival, and then return to zero after the arrival. In each of the test runs, the data window was chosen such that the Pg arrival is located at the 20-second mark. This appears to be consistent with the algorithm output for each of the events. It is important to note in the figures above that there is a certain amount of startup time for each of the algorithms. The kurtosis algorithm has a startup time of 2.56 seconds, which corresponds to the window of data that is used to calculate each kurtosis value. Similarly, the STA/LTA algorithm has a startup time of 11 seconds (10 seconds for the LTA and 1 second for the STA) and ARC has a startup time of only 4 seconds. The effect of the algorithm startup is visible in the initial outputs of the neural networks.

The kurtosis and STA/LTA algorithms both appear to have clean outputs with fairly obvious indicators about the Pg arrival at 20 seconds. They each exhibit a few additional spikes in the neural network output. The ARC output, however, is very poor in comparison. Although the ARC algorithm appears to present strong indicators of the presence of a signal, it is very impulse throughout the signal. These other impulses are simply not visible due to the relative waveform scaling.

Multiple algorithm neural networks

We next formed a neural network using multiple algorithms as inputs. Based upon the test results from the single algorithm networks, we used both the kurtosis and STA/LTA, but not ARC, algorithms. The network configuration is the same 2 layers and 13 nodes per layer as in the previous section. The only change from before is that the multiple algorithm network will have 13 inputs per algorithm for a total of 26 input nodes. The input sequences and their corresponding outputs are shown in Figure 8.

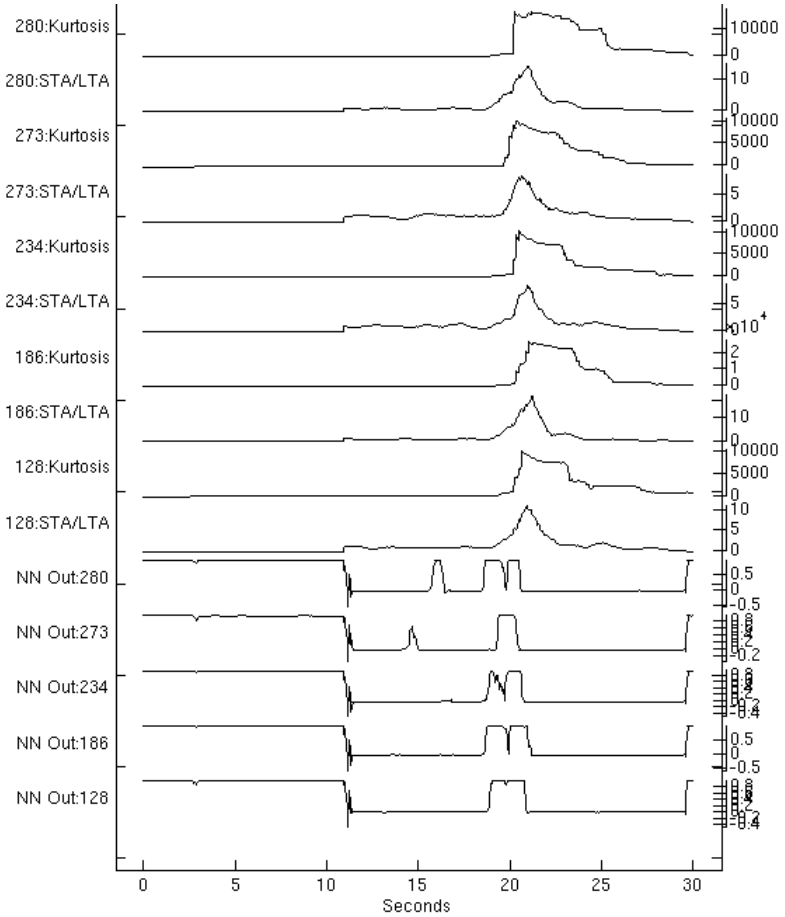

Figure 8. Kurtosis and STA/LTA processed data and neural network outputs for the data shown in Figure 4 using a multiple algorithm neural network.

Disregarding the algorithm startup, the combination of the two algorithms appears to have eliminated several of the false alarms that each of the single algorithm neural networks were exhibiting. Those output spikes that remain are the ones that were shared between the kurtosis and STA/LTA neural networks. This would tend to confirm that using multiple algorithms yields an improvement in the ability of a neural network to detect seismic events.

CONCLUSIONS AND RECOMMENDATIONS

In the course of evaluating the use of neural networks for use in seismic event detection, we have found that there are several key issues that need to be addressed prior to obtaining meaningful results. First, it is important to properly identify the features to be detected. We were able to largely resolve this problem by repicking the data. However, this issue could also be addressed by increasing the window of data that is fed into the neural network or by providing some means of manually adjusting the window for each training event. The second issue is the scaling of the window of data to be inputted to the neural network. Depending upon the algorithm, each window of data will need to be scaled independently to normalize for variation in SNR between events.

Using multiple algorithms in combination with a neural network should help to improve the performance of seismic event detection. We will be continuing in our study of this topic to further develop the ability to provide detection triggers from the neural network as well as examining additional algorithms for use in event detection.

REFERENCES

Jurkevics, A. (1988), Polarization Analysis of Three-Component Array Data, *Bull. Seism. Soc. Am*., 78, 1725-1743.

Savvaidis, A., C. Papazachos, P. Soupios, O. Galanis, N. Grammalidis, Ch. Saragiotis, L, Hadjileontiadis, and S. Panas (2002), Implementation of Additional Seismological Software for the Determination of Earthquake Parameters Based on MatSeis and an Automatic Phase-detector Algorithm, *Seism. Res. Lett.*, 73, 56-69.

Wang, J. and T. Teng (1997), Identification and Picking of S Phase Using an Artificial Neural Network, *Bull. Seism. Soc. Am*., 87, 1140 -1149.

Withers, M., R. Aster, C. Young, J. Beiriger, M. Harris, S. Moore, and J. Trujillo (1998), A Comparison of Select Trigger Algorithms for Automated Global Seismic Phase and Event Detection, *Bull. Seis. Soc. Am.*, 88, 95-106.

Zhao, Y. and K. Takano (1999), An Artificial Neural Network Approach for Broadband Seismic Phase Picking, *Bull. Seism. Soc. Am*., 89, 670-680.