Audio Noise Reduction Using Artificial Neural Networks

Abram Haich

**Abstract:** The use of Artificial Neural Networks (ANNs) in the past several years has increased drastically with the advent of more processing power and the unprecedented availability of data. One excellent area to apply ANNs is in the area of audio signal processing and pre-processing. Audio signal preprocessing is extremely important in all speech interpretation systems as well as in the production of audio-related work. This paper describes an intelligent system that uses ANNs to remove noise from speech signals. Additionally, this paper defines how the input data is translated into the neural network in pre- and post- processing.

1. **INTRODUCTION:**

In the field of audio signal processing, the process of filtering consists of a continuous mapping from a noisy signal (input) to an un-noisy signal (output). Typically, this filtering involves manipulating certain characteristics within a signal (i.e., volume, pitch, frequencies, etc.) in order to extract desired wave forms from undesired forms. In these types of situations, a simple function from a noisy to an un-noisy signal does not exist or it may exist but is unknown. It would therefore be of interest to determine this unknown function by using given noisy signals as well as their non-noisy counterpart. It is for this reason why Artificial Neural Networks (ANNs) are finding more usage in many various noise reduction problems [1, 2, 3, 4]. With the main goal of these networks being a mapping from noisy input data to clean non-noisy output data. These neural networks can be trained using a backpropagation algorithm and typically involve a regression type problem since a function is desired in most situations.

In this paper, an ANN was employed to solve the mapping from noisy audio input data to a non-noisy output. This ANN was trained using speech datasets and noise datasets where the noise files consisted of various background noise (public location noises, telephones, wind, etc) and the speech files contained different genders and words. The ANN model was trained using the famous backpropagation algorithm and consisted of several different layers each with a specific number of input and output neurons. This network was expected to learn and generalize specific characteristics about the noise in order to correctly map any untrained noisy signals to their corresponding un-noisy counterparts. Layers used in the model consist of:

* Long Short Term Memory (LSTM)
* Embedding
* Activation

1. **ARTIFICIAL NEURAL NETWORKS BACKGROUND:**

The process of building an ANN consists of two parts: learning and testing. Learning consists of tuning weights within the NN in response to particular patterns found within the input layer. How the weights adapt to the particular input and output examples within a training set is an example of a training algorithm. The process of testing a given ANN consists of providing an input signal to the input layer and verifying a response at the output layer.

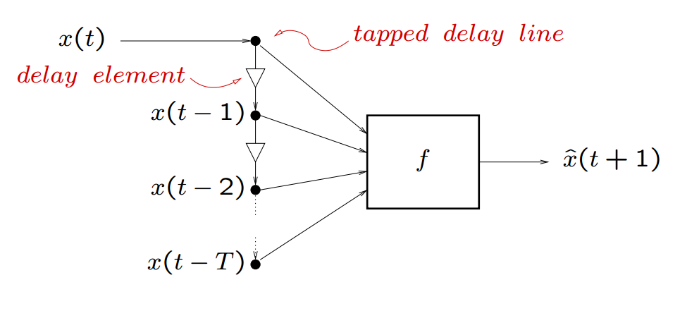
When it comes to designing an ANN the architecture for a particular application will depend on the size of the network (number of layers, number of input and output neurons), the training algorithm employed as well as the types of layers used. We will now proceed to cover the layers used within this paper.

* 1. **Long Short Term Memory Layers**

Long Short Term Memory (LSTM) is a specialized type of a recurrent neural network (RNN) architecture that is more accurate at determining patterns in time sequential data over conventional RNNs [1]. RNNs are essentially a network layer that contains connections with cycles within them. This cycle allows them to work very well with time series data since it allows them to remember past data. However in practice RNNs are unable to connect certain patterns beyond a certain point in the past. If a pattern occurs a long duration in the past the network does not necessarily relate the historic one to one in the present. An LSTM solves these shortcomings by introducing something called memory blocks within the RNN hidden layer [1]. These memory blocks allow the network to store a temporal state of the network in addition to multiplicative units called gates to control the flow of information. Each of these memory blocks control the input gates as well as the output gates of the network allowing the network move information. The ANNs in this paper use three layers of LSTMs for use in remembering the patterns of noise within the audio time series data.

* 1. **Embedding Layers**

Embedding layers are used to transform one-dimensional time vectors into infinite-dimensional spatial vectors. They allow a simple transformation of historic data into a fixed length (T) called an embedding dimension. Using embedding layers allows past values of a time series input space to be used as a predictor of future series data. In Figure 1 we described an input vector x at time t being fed into a function f used to predict the next time value t + 1. These layers are especially useful as inputs into an LSTM layer as they provide a fixed amount of historic data for an LSTM layers memory blocks.



* 1. **Activation Layers**

An activation layer is used to describe the output of a particular layer within a network. These can take many forms such as a simple binary system: true (1) and false (0), or they can be as complex as nonlinear functions such as tangents and sinusoidal waveforms. Since the content contained within our audio signals is sinusoidal in nature we used an activation function of sigmoid within our network as the final layer.

1. **NETWORK ARCHITECTURE:**
   1. **Dataset Creation**

Generation of the neural network training data was accomplished using Python to overlay noisy data onto speech signals. The noise signals were repeated to match the duration of the speech signals and then overlaid into the background of the speech signals. Noisy data consisted of public location background noise, telephones, wind, construction noise and many other variations of these. Over 2000 training examples were generated using 9 speech files and 101 noise files. The mapping of noise and speech signals were maintained using a simple xml file.

* 1. **Data Normalization**

The process of building an ANN for audio signal noise reduction is a twofold process: the first step is to normalize the data so that all of the signals are all within some relativistic scale and the second is to interrupt this normalized data using a specially designed network in order to predict some normalized output. For normalizing data a simple algebra calculation was performed (Equation 1). This normalization forced the audio signals to decimal values between the range of [0,1]. After normalization of the data, network architecture construction was performed.

Equation 1

* 1. **Network Architecture**

Construction of the ANN was done using multiple ANN layers within the Keras Python toolkit. The architecture consisted of an Embedding layer, three LSTM layers, Time Distributed density layer, Time Distributed Merge layer and lastly an activation layer. 256 input neurons were chosen as the input vector used to predict a 256 output vector. The time axis was built using the embedding layer as a medium between the input layer and the LSTM layers. Samples were wav files where the sample rate was 200 Hz. Neurons between LSTM layers were made to be twice the input layers (512) in order to account for the time axis output from the embedding layer. In order to reconstruct a wav file the output from the stacked LSTM layers both a Time Distributed density layer and a Time Distributed Merge layers were used to bring units back down in dimensions (specific feature of Keras). An activation layer of sigmoid was chosen after the previous layers since the underlying signals were sinusoidal in nature.





* 1. **Network Training**

Training the network consisted of using the Back Propagation Algorithm [5] to tweak the weights to match the output signal. All wav files were divided into 90% training and 10% testing. The routine repeated training until the mean squared error (MSE) between the input and output signals was less than a specified amount or until a max amount of iterations were ran.

1. **EXPERIMENTAL RESULTS:**

In this section the results obtained after training the ANN on 90% of the dataset are discussed as well as issues overcome and improvements made to the network after and during training and implementation. The first thing to note is that in order to monitor network progress 10% of every input signal was used to validate the network during the training process. At the beginning of training the MSE between the input and output signals of the ANN were as high as 0.99 (range of MSE is [0,1] due to range on input signals). After training, if the noise signal was of a simple waveform (i.e. simpler noise data i.e. not public noise but telephone noise) the MSE at was lower than 0.0002 (*Figure 3*). However if the noise signal was not easily distinguishable (something like public background noise) the MSE could go as high as 0.65 after training the network.

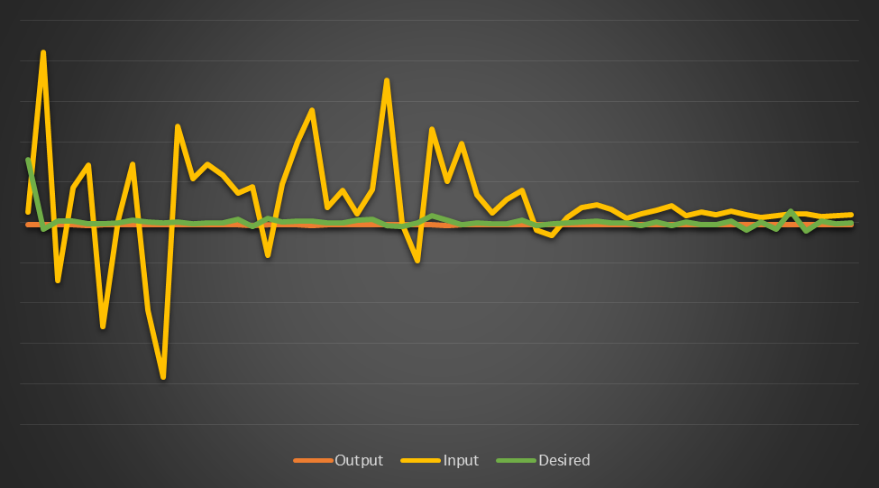


Figure 3: MSE < 0.0002

There were several periods during training in which the network was re-designed due to some shortcoming found. For example, the first implementation of the network did not use normalization and had major problems generalizing it’s representation of noise. This was later fixed by adding normalization to both the input and output signals. Some other problems overcome were:

* Input and output sizes:
  + The input and output sizes were later found to be more effective if they were larger in size. So after figuring this out a size of 256 was used in the final network since this is a multiple of 2 that fits within the memory space of the computer program.
* Type of moving window to use:
  + When first starting out a moving window was used where one value would be different and all the others would just shift within that window (i.e. [x0,x1]=>[x1, x2]).
* Representation of sound:
  + Due to all of the research done with Fast Fourier Transforms (FFTs) within the field of audio signal processing they were later implemented within our ANN. An FFT was used on input signals and an Inverse FFT (IFFT) was performed on the output data. It was later determined that this made the results of the ANN measurably more accurate than without them since they allowed a transformation to the frequency domain where noise is more distinguishable over non-noise.
* Representation of the time dimension:
  + Before the use of embedding layers a simple time dimension was added within the input array to make it a 2-dimensional vector of time and sound magnitude. It was later determined that embedding layers did a better job at keeping past data then use of a time dimension and so this implementation was scrapped.

Even though the above issues were addressed there was still some that were not due to time-constraints. So these problems are still present within the model. The first was a problem introduced by using an FFT in an LSTM architecture. Remember that an FFT converts a real sinusoidal signal from a time domain to a frequency domain signal. When using this with an LSTM there were some situations where a signal would get latched due to lower and higher frequencies converging (*Figure 4*)

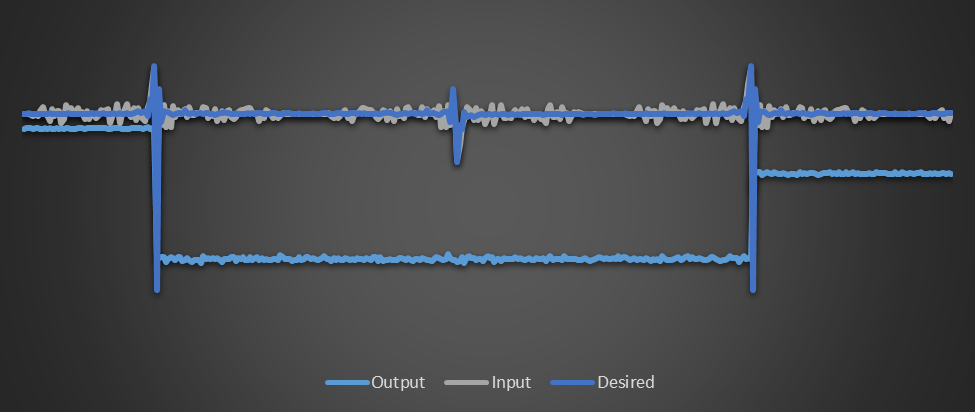


Figure 4: LSTM Latch

This issue could potentially be fixed by introducing some Feed Forward Layers at the top for the FFT and then somewhere in the middle layer a transformation could be to convert the signal back to a time signal before feeding into the LSTM layers. Another issue present in the model is the fundamental nature of the input and output signals. The accuracy of the model could be drastically improved by instead of predicting a signal without noise, the noise itself was predicted. This would allow the model to pull out the noise signal instead of the underlying desired signal therefore making it more portable to other platforms in addition to improving accuracy and precision.

1. **CONCLUSION:**

In this paper an ANN was discussed that was able to generalize on audio noise data. This generalization allowed the removal of noise from a noisy signal in order to extract an underlying speech signal. Performance of the ANN was very accurate with simple noise data but could be inaccurate if the noisy data is hard to distinguish from the underling speech signal. With a few improvements a lot of the issues present in the current ANN could be solved and a very powerful noise reduction ANN could be built.

**References:**

| [1] | O. N. A. AL-Allaf, "Removing Noise from Speech Signals Using Different Approaches of Artificial Neural Networks," *I.J. Information Technology and Computer Science,* pp. 1-11, 2015. |
| --- | --- |
| [2] | S. S. S. K. Vartika Anand, "Intelligent Adaptive Filtering For Noise," *International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering,* vol. 2, no. 5, p. 1, 2013. |
| [3] | A. Uncini, "Audio signal processing by neural networks," *Neurocomputing,* vol. 18, no. 55, p. 1, 2003. |
| [4] | A. S. F. B. Has¸im Sak, "Long Short-Term Memory Recurrent Neural Network Architectures," *INTERSPEECH,* p. 1, 2014 . |
| [5] | D. T. a. K. Laskowsk, *Neural Networks for Time Series,* Pittsburgh: Carnegie Mellon University, 2006. |
| [6] | R. Rojas, *Neural Networks: The Backpropagation Algorithm,* Berlin: Springer-Verlag, 1996. |