ECSE 421: Embedded System Project

Milestone 2: Learn Music

Mingrui Zhang, Yufei Liu, Jun Wen, Szu-Chieh(Jeffrey) Fang
Department of Electrical and Computer Engineering
McGill University
Montreal, Quebec, Canada

***Abstract*— An Embedded system is a computer system with a dedicated function within a large mechanical or electrical system, which can solve real-time problems with the help of data detected by sensors from the physical world. Artificial neural network is one of the most popular ways to implement such a system efficiently. It is also a common method used in machine learning. The machine part of the neural network is a large collection of basic processing elements called neurons, inspired by real neurons in human nervous system. The learning part is a set of automatically updated databases; it affects the final outputs of the system and helps derive more accurate results. This project focuses on building a neural network which is capable of recognizing sound intervals using LabVIEW. Different designs and implementations are considered and tested by evaluating several parameters, including performance, execution time, and code efficiency. Consequently, an array input, which is fed into a looping structure with two neuron layers, can generate a relatively more accurate result compared with other approaches. This report presents and discusses the final design involving testing, evaluation, and comparisons with other alternatives.**

I. INTRODUCTION

This project consists of a neural network built on a myRIO board using LabVIEW. By utilizing the field-programmable logic array (FPGA) component inside the board, our program can process sound data received from the Audio/In port and recognize sound intervals. This recognition requires two stages: "training mode" and "inference mode". The training mode is the stage where the system learns. It treats user-required output as an input that can correctly indicate the possibility of what a given sound interval is. Then, using the processed data from the audio input, the system compares the required output with the self-generated one to update necessary parameters and produce a new output closer to the desired one. In the inference mode, our system simply generates outputs based on the sound intervals that the network has learned before using the most recent parameters. In addition, the neural network is made up of two parts, the feedforward network and the backpropagation algorithm.

II. SYSTEM ARCHITECTURE

A. Overview

The system contains four layers, including an input layer, two neural layers, or "hidden layers", and an output layer. They are implemented in a flat sequence structure in LabVIEW. The input layer takes the audio sound inputs generated by another myRIO board and converts them from analog to digital signals, which will finally produce a set of discrete numbers in a form of a one-dimensional array. This array, as an input of the fast Fourier transform (FFT) spectrum, is then further processed to derive the array indices of specific data points which represent the amplitude peaks. Finally, the system reforms these array indices into a new one-dimensional array and sends it as an input of the neural network. The above structure of audio input processing is shown in Fig. 2.1.1.

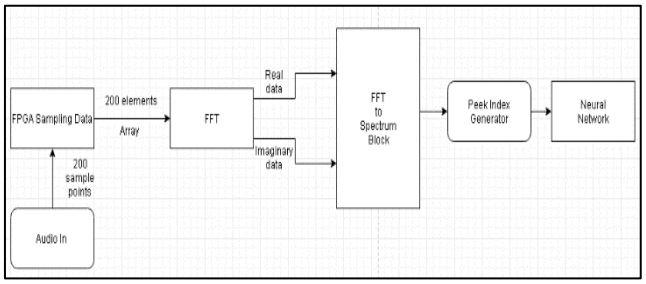

Fig. 2.1.1: Structure of Audio Input Processing

The neural network is located inside the two hidden layers. Each hidden layer has two neurons, and each neuron requires two discrete data inputs and two special parameters, or "weights". The input array is looped two times to extract desired input data, and each extracted input is fed into two neuron units along with two different weights, as depicted in Fig. 2.1.2.

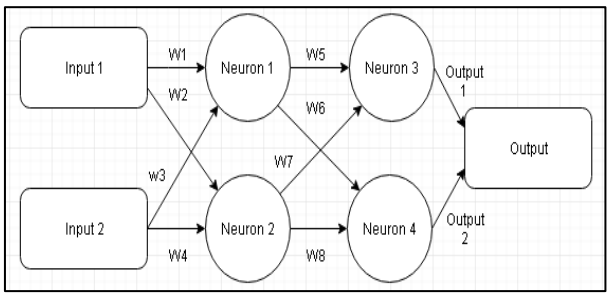

Fig. 2.1.2: Basic Structure of the Neural Network

Therefore, we have totally four different weights from the input layer to the first hidden layer, and values of the weights are extracted from the weight matrix stored in the system and can be updated depending on mode selections. Same thing also happens between the first and second hidden layers, where another four different weights are derived from the

weight matrix together with the outputs of the first hidden layer. These processed data are treated as the inputs of the second layer. Consequently, two final discrete data outputs coming from the second layer are then fed into the output layer, which will generate outputs within a range from 0 to 1. The error calculation algorithm, using the actual and expected outputs, also runs in the same layer. Figure. 2.1.3 shows the overall logic of how we combine the weight blocks and neutrons.

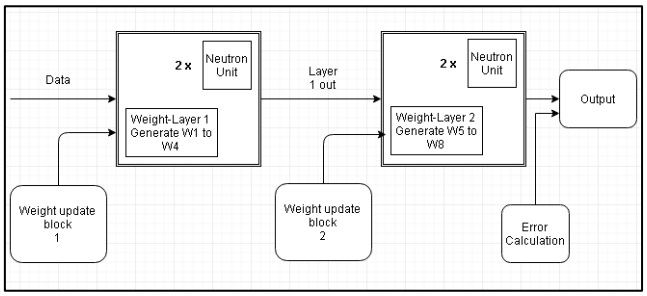

Fig. 2.1.3: General Neural Network Block Diagram

B. *Components*

1) *Audio input processing block*

Inside the FPGA component, the system samples 200 audio input data, and they are automatically stored in an array using a for-loop. The reason we choose 200 as the parameter is that after 200 times of sampling data, the results are close to the ideal ones without sacrificing too many resources and efficiencies (running time) within FPGA. Later, the data array is fed into a FFT block outside FPGA, and the block will transform the sampled sound data from time domain to frequency domain. Next, the processed data is sent to the FFT-To-Spectrum block to generate a new array that can be read by a Waveform Chart in LabVIEW. From the graph shown in the chart, users can monitor how many sound frequencies the input has. For example, if the system receives a two-note sound interval which contains two different frequencies, the graph will have two peaks where each peak represents a distinct frequency, as shown in Fig 2.2.1. After that, these data are further proceeded to collect array indices where peak values present. The indices are stored into an array and passed to the next sequence.

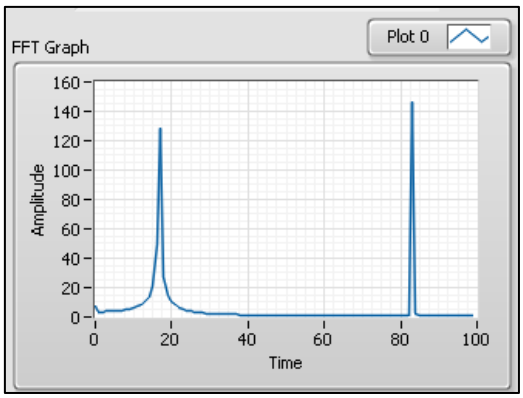

Fig 2.2.1: Front Panel FFT Graph

2) *Hidden layer*

After receiving input signals from the audio input processing block, the data now has the form that the neural system can operate as we expect. The structures where the input data is processed are the "hidden layers". They are consisted as several small processing units called "neurons", and these neurons are used to receive, manipulate and, output new data. A single neuron may be powerless. However, as the quantity of units magnifies, and the linkages between them become more complex, the ability to realize the machine learning system reveals.

The neuron unit in our system is implemented by constructing a simple weighted linear combination, which simplifies the required tasks, inside a sigmoid function, as shown in the equation (in vector form) below.

$$output = \frac{1}{1 + \exp(-(x^T w + b))}$$

It is called the "activation function". In the equation, x is the input vector. w is the weight vector containing weights corresponding to each input, and b is the bias, which is set by users in our design. The purpose of using such a function is that its result is bounded between 0 and 1 [1], and this normalizes the output of the neuron, mitigating possible extra works in later parts.

The realization of a neuron unit in LabVIEW is shown in the figure below (Fig. 2.2.2). We use vector input blocks which are two-dimensional arrays to represent the input data and weights. The reason of using these blocks rather than several scalar blocks is that if we need to extend the quantity of input/weights in the future, we only need to add extra parameters into the vectors. In addition, our way to increase the number of neurons is using nested for-loops, which can easily generate indices to manipulate multidimensional arrays. The for-loop index N represents the number of neurons that a layer has.

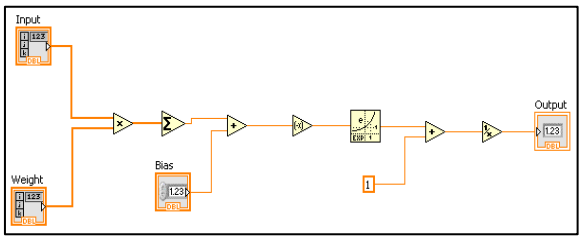

Fig 2.2.2: LabVIEW Block Diagram of the Neuron Unit

We also place an Array to Cluster block outside the outer-most for-loop to ensure that it does not affect the format of the input array. In each outer-most for-loop, weights are extracted from a two-dimensional array and reformed into a one-dimensional array to feed into a neuron. The computed outputs for each neuron are also concatenated to one-dimensional array and passed to the next hidden layer. One thing to note is that the structure of the second hidden layer is the same as the first hidden layer.

The audio input processing block and hidden layers together form the feedforward network as defined.

3) Error calculation block

The implementation of the error calculation in our system utilizes the concept of mean squared error, as shown in the following equation:

$$\text{error} = \sum_n E_n = \sum_n \frac{1}{2}(t_n - a_n)^2$$

In this equation, E_n is the error from each output. t_n is the expected output, and a_n is the actual output.

In LabVIEW, the actual and expected outputs are represented in the form of vector blocks, with the same consideration mentioned above. The error calculation block takes in the expected outputs entered by users and the actual outputs from the second hidden layer to produce its result to an indicator on the front panel (user interface). The computed errors can then be utilized to update the weights of neurons, executing the learning process. This is the start of the backpropagation algorithm.

4) Weight updating block

The weight updating block, as part of the backpropagation algorithm, calculates new weights relying on the following weight updating equations

$$W_{ij} = W_{ij} - \acute{\eta}\frac{\partial E}{\partial W_{ij}}$$

$$\frac{\partial E}{\partial W_{ij}} = \frac{\partial E}{\partial out_j}\frac{\partial out_j}{\partial net_j}\frac{\partial net_j}{\partial w_{ij}}$$

where W_{ij} represents the weights between neuron i and neuron j, $\acute{\eta}$ indicates the user-defined learning rate, E represents the error, net_j is the weighted sum of inputs of neuron n, and out_j is the output of the activation function for neuron n. As we can see from the equations, the values of the partial differential equations may depend on the previous layer. Therefore, different layers will have different weight updating blocks.

In the first layer $\frac{\partial E}{\partial out_j}$ does not depend on any other neurons. However, the value of $\frac{\partial E}{\partial out_j}$ in the second layer has dependency on the first layer. Thus, the implementation of the weight updating method of the second layer will change. These changes will not directly reflect to the weight array. The updated weights are stored in temporary arrays instead.

5) Mode selection block

To select from different modes, a Boolean which is decided by a button on the front panel is utilized. Depending on whether the button is pressed or not, two selectors will decide whether to replace the original weights with the values stored in the temporary storage arrays mentioned above. The system will always stay in the inference mode if the button is not pressed.

Error calculation blocks, weight updating blocks, and the mode selection block form the backpropagation algorithm of our system.

C. User interface

Users can interact with the system through the LabVIEW front panel, which includes two main parts: the Reading Panel and the Training Panel. In the Reading Panel, there is a numerical display indicates the data the system receives. In order to obtain the raw fast Fourier transform results of the inputs, a separate numerical display and a graphical display are also placed in the Reading Panel. The data are exhibited in the form of Amplitude-Time graph as shown in Fig 2.2.1. The remaining parts of the Reading Panel include indicators reflecting the current states of the hidden layers. In the Training Panel, there are numerical displays showing errors, required outputs, and updated weights in the hidden layers. The Training Panel also contains a button for users to switch between modes.

Fig 2.3.1 shows an overview of how the feedforward network, backpropagation network, and user interface interact with one another.

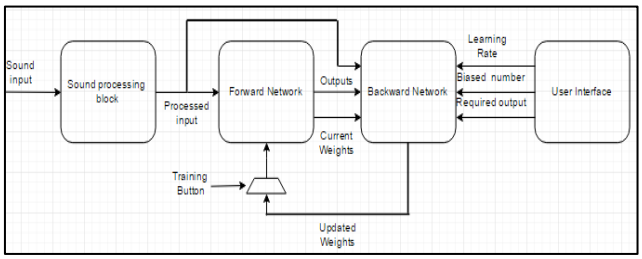

Fig 2.3.1 Stage Interactions

III. SYSTEM FEATURES, TESTING AND EVALUATIONS

A. System features

Our system utilizes a flat sequence structure, which ensures that each sub-diagram in it executes accordingly. Also, data leave a sub-diagram only when it finishes execution, and this makes sure that the data are only processed in the components within the same sub-diagram, avoiding racing conditions. In addition, since frames in a flat sequence structure execute in order from left to right, we can easily understand how data flow. Moreover, as this flat sequence is located inside a while loop, the system can keep running the whole sequence until the user launches a stop order. Another point is that for-loops are utilized at many places in the system with multiple purposes. They can help minimize the number of required blocks, such as the neuron units, and their loop indices also assist to generate array indices to extract data from weight storage arrays.

B. Testing

We utilized the test-driven development method to ensure that every block of our system works both independently and globally as expected. For the audio input processing block, amplitude spectrum is used (as shown in left-top corner of Fig 2.3.1) as an indicator to show that the input data have been transformed to the correct form. For example, when we used a two-note interval as an input, there were indeed two peaks in the amplitude spectrum indicator.

For the neuron unit, we did not test it specifically since it is just an implementation of math equations. Instead, we tested it within the whole feedforward network. At the beginning of the test, the expected output should be 0.5 since no weight value was inserted. After we verified that the actual output

was 0.5, we then moved to the next step and implemented the backpropagation algorithm. Before we started the final testing, we connected several indicators to the hidden layer 1, hidden layer 2, weight update block of hidden layer 1, and weight update block of hidden layer 2 to have a better understanding of what was happening inside the system.

When it came to the integration testing, we used two myRIO boards. myRIO board 1 was loaded with project 1 – make music, which played a music interval as the input to myRIO board 2. We started testing by playing only one note C4 and setting the expected output to be $(1,0)$, trained the system for a while until the actual output was stable, and then played another note B4 with the expected output $(0,1)$. After the training was finished, we switched to the inference mode. When playing C4, the system should recognize it and give an output of $(1,0)$. Similarly, when playing B4, the output should be $(0,1)$. Once the system functioned correctly for one note, we then proceeded to test the two-note interval with the same principle.

C. Final Evaluation

The system worked successfully for two-note interval input as well. At the beginning before we started playing any sound, the output was $[0.5, 0.5]$ as expected (Fig 3.3.1).

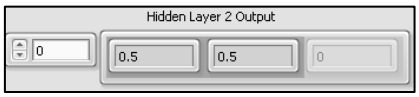

Fig 3.3.1: Initial Output before Training

During the test, we first gave a two-note interval which contained notes A6 and C3 and then set the expected output to $[1, 0]$. We trained the system until the first output was closed to 0.8, rather than 0.9 or 1.0, to avoid overtraining and then released the training button. After that, we gave a second two-note interval which contained note B2 and A5, set the required output to be $[0, 1]$, and repeated the previous training procedure. After the training was completed, the system then switched to the inference mode. When playing the first two-note interval, the ideal output should be $[1, 0]$ to indicate that the system could recognize this sound. As shown in Fig 3.3.2, the real output was approximately $[0.668,0.314]$, which was not perfectly closed to 0.8, but acceptable.

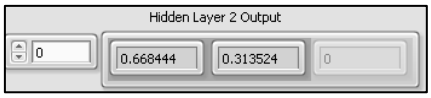

Fig 3.3.2: Real Output when Playing the First Interval

When playing the second two-note interval, the ideal output should be $[0, 1]$, indicating that the system recognized the second note interval as well. As shown in Fig 3.3.3, the system gave an output of $[0.16, 0.828]$, which was a relative accurate result. One thing to note is that the bias number we choose for neurons in both layers was 0.05.

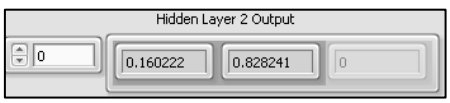

Fig 3.3.3: Real output when playing the second sound

There are still observable problems in the system. For example, it cannot distinguish two-note intervals in which the frequencies are too close since the array indices of the peaks do not have distinct differences. Another problem is that the quantity of outputs, which represents the number of note intervals that the system can identify is limited. For instance, in theory we can add more outputs to the system by placing more neurons in each layer based on our design. However, since we chose peak indices as our input data sets, the number of valid input data will be limited to the number of note frequencies can be in an interval. As more neurons are added, for-loop indices will also increase, which leads to using array components that are not valid. This means that we will eventually have the same number of outputs as the maximum number of frequencies can be contained in an interval.

D. Alternative design

To solve the first problem mentioned above, we considered using different numbers of layer. By adding additional layer, weight storage array is not biased too much for similar note intervals, and it also help avoid overtraining. However, the test result barely improved when the number of the hidden layers changed from two to three. Considering the code size, difficulty in implementing $\frac{\partial E}{\partial out_j}$ in LabVIEW, and time limit, we decided to stay with the two-hidden-layered design. In addition, adding more neurons within one hidden layer also produced similar results as the previous design.

For the second problem, we first considered using different input data sets. Instead of using the one that contains only peak indices, we tried using the whole FFT data, giving us enough input data. However, data of different frequencies hardly have distinguishability in reality. During the test, the system output was not sensitive enough to respond to the changes in the input data set, and it was proven that this design performs worse than our final design.

IV. CONCLUSION

It was a great experience to work on this project as it provided an opportunity to explore the fundamental knowledge of artificial neural network as well as the basic concepts of machine learning. Even though our system did not work perfectly, we still gained valuable experiences during the process. In the future, the improvements should focus on inputs and algorithms, and we recommend using more representable input data sets. Additionally, utilizing updated biased number instead of a user-inputted one may also solve the above problems, which is a new area we have not touched. For enhancement in the user interface, using the press button on myRIO board to select modes may be simpler than working with the front panel on screen.

REFERENCES

[1] Quoc V. Le. A Tutorial on Deep Learning. [Online]. Available: http://cs.stanford.edu/~quocle/tutorial1.pdf. [Accessed 5 April 2017]