FINAL PROJECT: NEURAL NETWORKS FOR EMBEDDED SYSTEMS

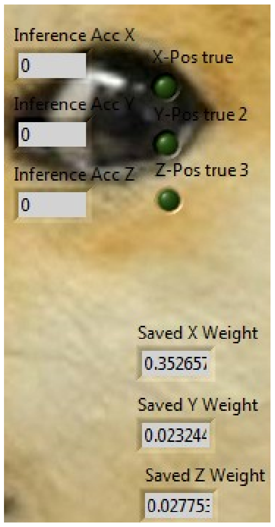
April 11, 2016

Authors
Muhammad Hussain
Sean Ryan
Ahmad Aziz

Professor
Brett H. Meyer

Abstract: This paper presents the design specifications and layout of a neural network implemented on an FPGA and a microprocessor which can learn to recognize spatial orientations taught to it by the user. The neural network basically uses machine learning to indicate the coordinates when the board is held in different positions. The system has two modes: *Training mode* and *Inference mode.* In the training mode, the user provides the system with a "correct output" parameter and then holds the board in a certain position, allowing the network to learn the appropriate parameters so that it can give the correct output when held in that position in the future. In inference mode, the system outputs whichever pattern it detects out of the positions the network has learned from the user.

Introduction: This project is aimed at putting a neural network on a myRIO device and programming it to learn to recognize spatial orientations taught to it by the user [1]. The front panel of the instrument consists of a virtual button that switches between training and inference modes. The true weights are the target weights, 'Etta' is the learning rate and X, Y, Z_{Old} are the random initialization weights before learning occurs. There are many indicators; X, Y, Z_{New} are the transfer function outputs in g's, X,Y,Z positions show the true position of the NI device from a base reference point already stored by the manufacturer in the device. For the inference mode, the acceleration indicators in X, Y, Z directions show the true device acceleration in g's.


Inference Acc X
0
X-Pos true
Inference Acc Y
0
Y-Pos true 2
Inference Acc Z
0
Z-Pos true 3
Saved X Weight
0.352657
Saved Y Weight
0.023244
Saved Z Weight
0.027753

The saved weights are taken from the training mode after learning a new position. The LEDs shown on the VI light up when either of the axis are at the learnt position from the inference. The inference calculates the error between the true acceleration and the learnt position. If the error factor is less than 0.4, the LEDs light up. If not, they remain dark.

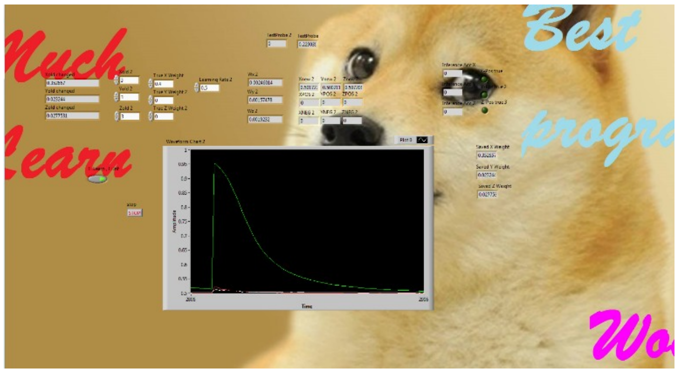


Figure 1 - Front panel for the neural network VI (virtual instrument)

Part 1: Feed forward Network

The feed forward network consists of a network of collection of artificial neurons that mimic the action of biological neurons by multiplying each input signal by a weight. These neurons are connected in layers with no feedback loops. Our design uses a single layer neural network with 3 active neurons that are interconnected to each other. The weighted signals (W1, W2 and W3 shown in *figure 2*) are summed up and passed through an activation function or transfer function. The transfer function is implemented using the following equation:

$$\text{Transfer Function} = \frac{e^x}{1+e^x}$$

The transfer function idea is taken from a Google report [2]. We used a logistic and a hyperbolic transfer function for a comparison analysis later in the design process. The following diagram illustrates the aforementioned implementation.

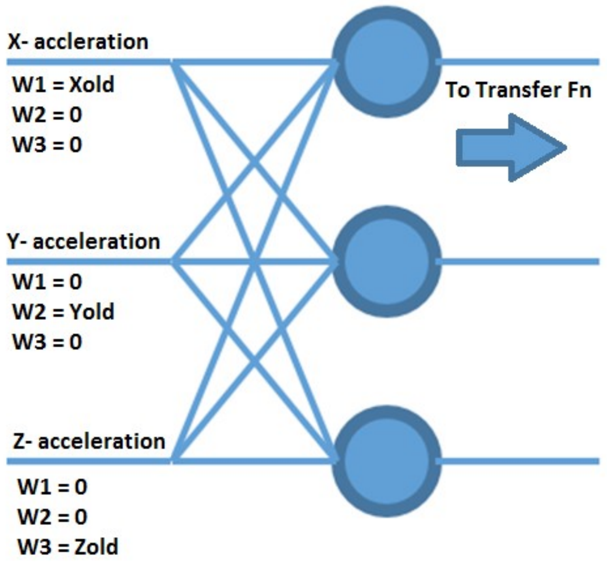


Figure 2 –Our Implementation of a single layer neuron layer [1].

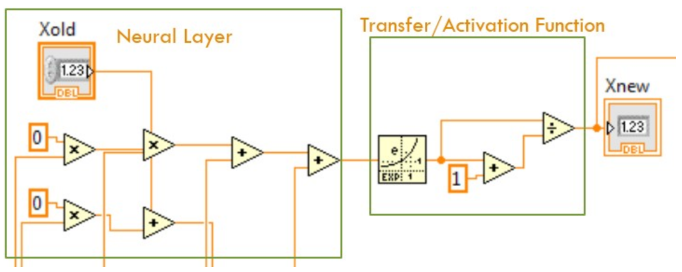


Figure 3 – Implementation of transfer function & neural layer

X-new is the output of the activation function for the X direction as can be seen in *figure 1*. Using similar implementation, outputs for all directions are calculated. Using the outputs of the activation function, the system determines whether the direction is in the positive or negative axis.

Activation function output	< 0.5	> 0.5	= 0.5
Direction	**Negative**	**Positive**	**0**

A waveform chart has been used to determine the magnitude of the acceleration. The acceleration sampling has been set to 100ms (10Hz).

Part 2: Back Propagation Algorithm

The purpose of the back propagation algorithm is to provide the artificial neural network with a mechanism to learn. The back propagation algorithm executes multiple tasks; runs the feed forward network on an input training example to compute the outputs of all the neurons, computes partial derivatives of the error, E and updates each weight in the network using the update equation [1]. This is repeated until the outputs are satisfactorily close to the target outputs. The weights for x, y and z are updated using the feedback equations as shown below:

$$W_{X \vee Updated} = X_{Old} - \eta [\frac{dE}{d_{wij}}]$$

$$W_{Y \vee Updated} = Y_{Old} - \eta [\frac{dE}{d_{wij}}]$$

$$W_{Z \vee Updated} = Z_{Old} - \eta [\frac{dE}{d_{wij}}]$$

Where $\frac{dE}{d_{wij}}$ is the differential in the error, E and η is the learning rate. A feedback node replaces the old value with the new value as shown in *figure 5*.

For the design of the neural network system, we changed the error equation from the one provided in the specification. An approximation is used rather than a true differential using calculus which makes the implementation in LabView easier. This can be seen in *figure 3* as we take the differential in the error using *equation 1* and dividing it by the differential in the old and the new values of the weights.

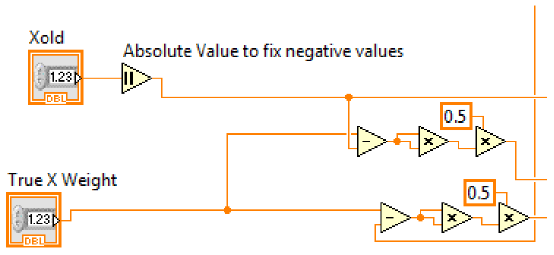


Figure 4 - Differential of error: $(\frac{dE}{d_{wij}})$

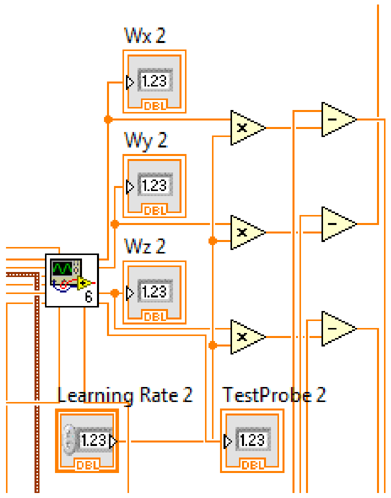


Figure 5 – Change in weight

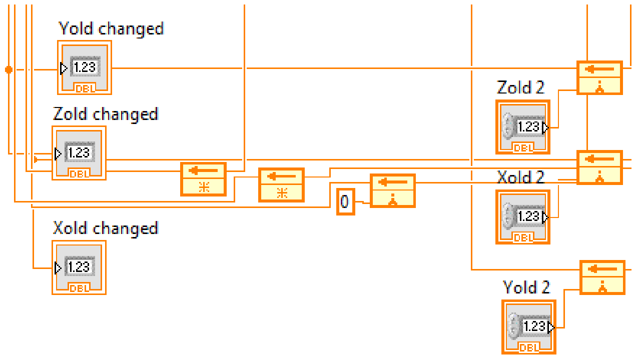


Figure 6 - Updated weight

The errors in the X, Y and Z directions are calculated using the following equation:

$$E_{X,Y,Z} = \frac{1}{2}(True\,Weight_{X,Y,Z} - X,Y,Z_{New})^2$$ - ***Equation 1***

The error equation is taken from the specification [1]. Where η is the learning rate, wij is the weight connecting neuron 'i' to neuron 'j' and $\frac{dE}{d_{wij}}$ is the partial derivative of the error with respect towij. Each of the partial derivatives is calculated using the chain rule:

$$\frac{d\,E_{X,Y,Z}}{d\,w_{ij}}=\frac{\frac{\frac{dE}{dout_j}*dout_j}{dnet_j}*dnet_j}{d_{wij}}$$

$$\frac{d\,E_{X,Y,Z}}{d_{wij}}=\frac{\Delta E}{\Delta(X,Y,Z)}*¿$$

$$\frac{E_{(X,Y,Z)_{Old}}}{(X,Y,Z)_{Old}}*\left[(X,Y,Z)_{New}*\left(1-(X,Y,Z)_{New}\right)\right]*(X,Y,Z)_{Old}$$

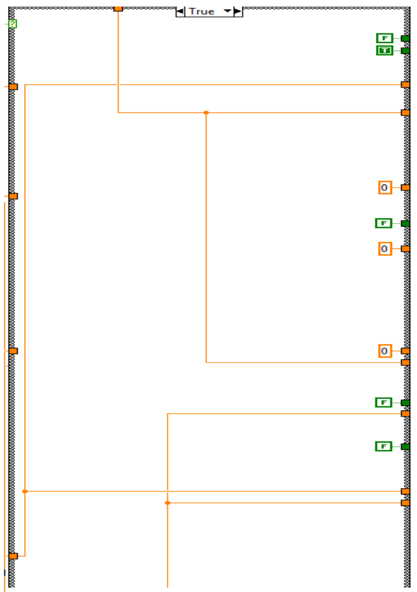


Figure - Inference mode case for neural network

Part 3: Complete Neural Network and Design Space Exploration

The complete neural network utilizes both the feed forward network and the back propagation algorithm to learn its relative position. The feed forward calculates the weights using the input signals and activation function. From here the weights are fed into the back propagation block of the system, which calculates the error and its derivative. When the output signals comes out of this block, they are multiplied by the learning rate of the system and the new weights are calculated by subtracting the old weight from this value. From here the new weights are fed back into the network and the process continues. Feedback nodes are utilized in the overall system to hold the old weights of x, y, and z until the system produces the error calculation, multiplied with the learning rate. When the new weight is calculated it is stored in the old value and the process continues. Additionally, old weights are fed into a waveform graph where they are displayed on the front panel for testing and verification. Furthermore, the switching between learning and inference mode is contained in this VI. A case structure identifies the two cases with true meaning learning mode is activated and false being inference mode. Depending on the values of the case structure different LED's on the board are it up. LED0 means the board is in learning mode and LED1 in inference mode. This mode can be switched using a button on the front panel VI. Within the false case of the case structure exists a series of blocks to normalize the error in the event that it is non-linear. This series of blocks performs *equation 2* (pg. 4) and compares it to 0.4. We choose 40% as a safe value to compare the error to as our design produces an error of 10% on average on a short learning time of$10\,seconds$. From here, important values such as old weights, new weights, and error are shown using indicators on the front panel VI.

Equation 2

$$\frac{\left|X,Y,Z_{InferenceAcceleration}-X,Y,Z_{SavedWeights}\right|}{X,Y,Z_{InferenceAcceleration}}\leq[0.4,0.1]$$

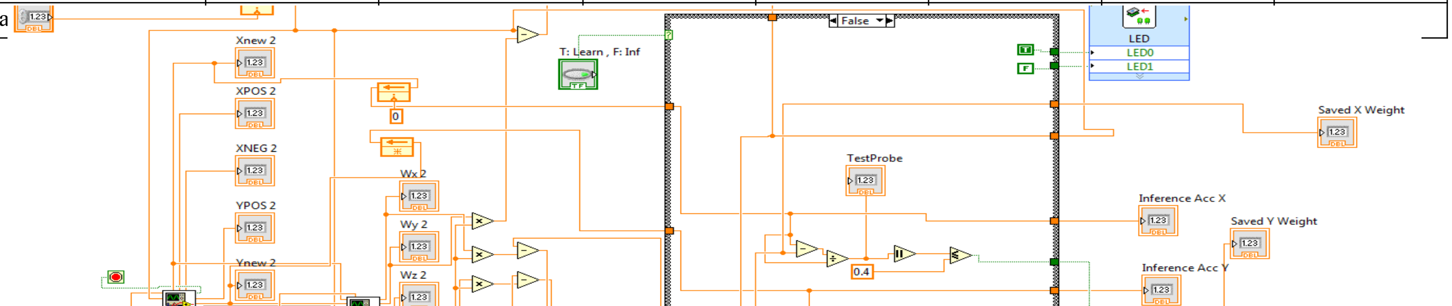
Part 4: testing and evaluation

Different learning rates were employed to test which learning rates were better in specific situations since the transfer function is non-linear. It was found that higher learning rates were better for lower true weights and vice versa. *Table 1* shows the tests for the Learning Rates.

	Inputs (in g)	Outputs (in g)	Learning Rates	Inference Accuracy Threshold
Max	8.0	0.72	10	0.50
Min	0.1	0	0.001	0.10

Table 1 - Parameters for VI use

	Front Panel Objects (K)	Block Diagram Objects (K)	Code (K)	Data (K)	Total (K)	Total Vi on Disk (K)	Code complexity
Original	49.7	131.2	30	62.4	273.2	60.4	1.1
Changed subvi6	49.7	130.7	30	62.4	272.7	60.4	1.1
a							



Ta
we
on
the
di
sp

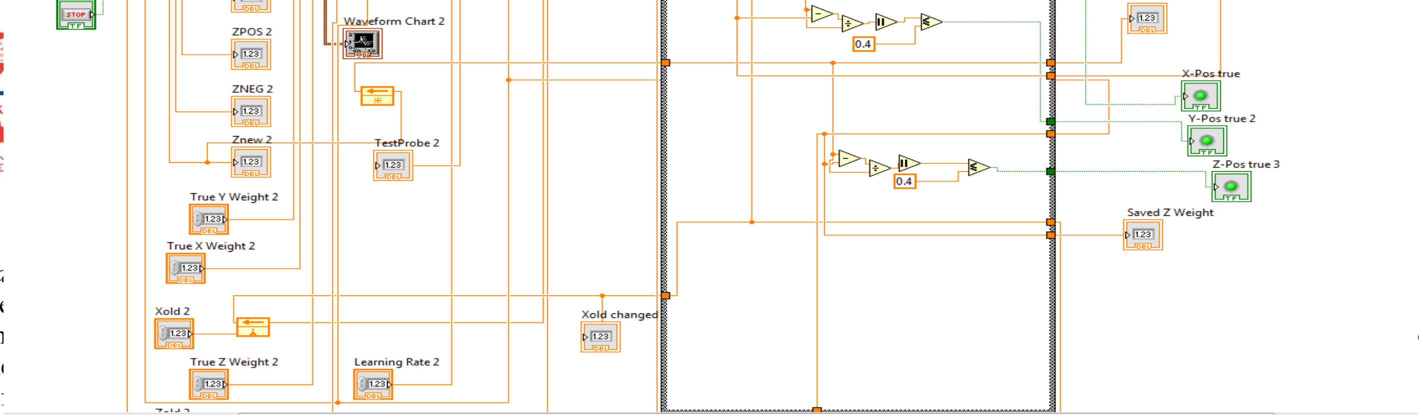


Figure - Learning mode for neural network

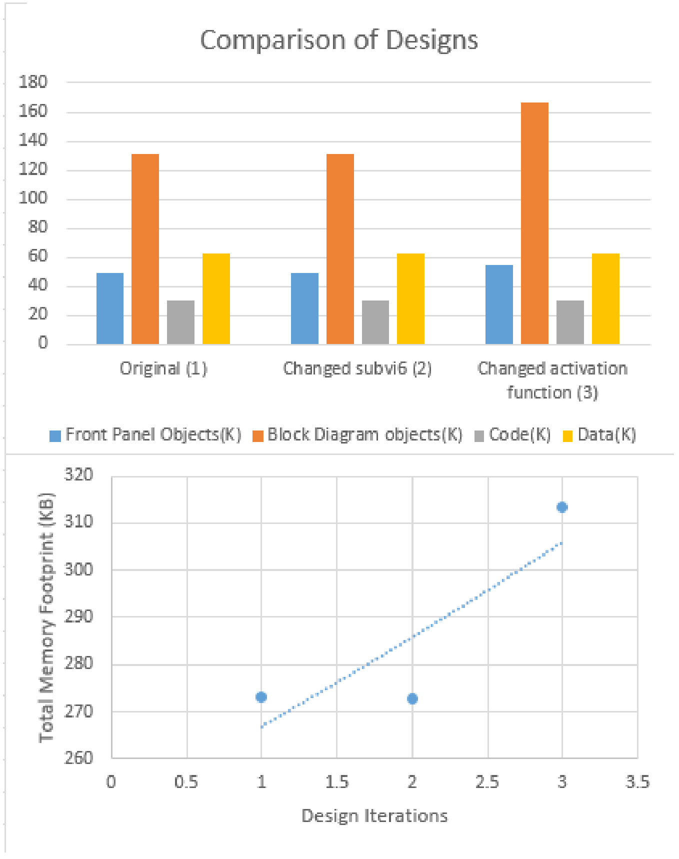


Figure 7 - Comparison of Design Alternatives with Memory in KB

Part 5: Project recommendations and conclusion

Given the opportunity to do the project over there is a few design aspects of the project we most likely would have done differently. The main aspect of the project that would be different is part 3, putting all components together to complete the neural network. Our completed neural network continuously runs the back propagation algorithm even when in it is in inference mode, which is very inefficient. A better design would be to separate the sub components so that training mode only computes when it is actually in training mode. Furthermore, we would have changed the case structure in our neural network, encapsulating the normalization of the error in a sub VI for a cleaner and more intuitive appearance. Lastly, the group would of attempted to create 2 levels of hidden neurons in order to have a more accurate representation of the data and minimize the number of connections in the network; Another justification of creating another hidden layer in the network would be decrease the expensiveness of computation and decrease the execution time as shallower

was a fun experience. The group learned to use LabView and had a dip into the world of real embedded systems for practical purposes.

References

[1] B. Meyer, "ECSE 421: Embedded Systems Project Neural Networks for Embedded Systems," Brett Meyer, Montreal, 2016.

[2] Q. V. Le, "A Tutorial on Deep Learning - Part 1: Nonlinear Classifiers and The Backpropagation Algorithm," Google Brain, Google Inc., Mountain View, CA, 2015.