Mixed Signal Circuits Optimized for Machine Learning

An Undergraduate Research Scholars Thesis

by

OLUWASEYI MORONFOYE

Submitted to the Undergraduate Research Scholars program at
Texas A&M University
in partial fulfillment of the requirements for the designation as an

UNDERGRADUATE RESEARCH SCHOLAR

Approved by Research Advisor: Dr. Samuel Palermo

May 2019

Major: Electrical Engineering

TABLE OF CONTENTS

ABSTRACT

Mixed Signal Circuits Optimized for Machine Learning

Oluwaseyi Moronfoye
Department of Electrical and Computer Engineering
Texas A&M University

Research Advisor: Dr. Samuel Palermo
Department of Electrical and Computer Engineering
Texas A&M University

The aim of this project is to develop customizable hardware that can perform Machine Learning tasks. Machine L earning is the science of leveraging advance statistics and data mining to "teach" computers how to recognize patterns and perform tasks without direct human instructions. Circuits optimized for machine learning using mixed-signal inputs will be able to act as a dedicated hardware for performing conventional computational tasks required in learning systems; improving both efficiency and power consumption.

The hardware will be comprised of an array of neurons, an activation-function block at the output of every neuron, and a back propagation protocol for every layer. The use of both digital and analog inputs will provide us with a means for not only faster computations, but also more intuitive results. The project will focus on answering the question: If and how we can implement an efficient circuit that uses both analog and digital inputs to train a device to learn patterns from data.

NOMENCLATURE

ANN	Artificial Neural Network
A	Amperes
GUI	Graphical User Interface
Hz	Hertz
kHz	Kilohertz (1,000 Hz)
LED	Light-emitting Diode
mA	Milliampere
MHz	Megahertz (1,000,000 Hz)
mW	Milliwatt
PCB	Printed Circuit Board
ReLU	Rectified Linear Unit
V	Volts
TP	True Positive
TN	True Negative
FP	False Positive
FN	False Negative

CHAPTER I

INTRODUCTION

Overview

The area of Machine Learning is now a substantial research area in this age of big data where humans are generating data with increasing frequency. However, the possibilities of integrating these machine learning capabilities into circuits are not as thoroughly researched. Embedded machine learning capabilities in mixed-signal circuits concentrate a majority of the computations to the sensor.

Near signal processing is more desirable in many cases than the cloud-based approach. Cloud-based computation is an approach that requires sending large amounts of data to servers (the cloud) to be processed and then sent back. This extensive data movement poses certain apparent limitations when speed is essential. Moreover, “The high dimensionality of raw data makes processing and transmissions very costly in terms of computational resources” [1].

This project aims to propose various circuit architectures that will support and improve embedded deep learning as well as the implementation of algorithms that support machine learning in the Analog signal domain.

CHAPTER II

LITERATURE REVIEW

This section will describe and discuss the past work done on integrated machine-learning systems. I will begin with a brief overview on the topic of machine learning specifically the learning model of an Artificial Neural Network, and then discuss past designs and implementations of hardware learning systems.

Artificial Neural Networks

Neural networks are an extremely robust learning system that is desirable for many of their properties such as self-organization and fault tolerance. They are able to detect and characterize patterns where closed form solutions might be too complex or might not exist. There has been an explosion of research on the topic of neural networks from the different benefits of various architectures (number and positions of the neurons in a network), to deciphering the complex sets of data in an Artificial Neural Network, ANN, that is used to make the predictions.

This sub section will provide a brief overview and simplification of the topic of ANNs. As the name implies neural networks are made up of a collections of computation cells called neurons/perceptrons. These neurons perform an extremely simple computational task, they take in a set of inputs $\in\{x_1, x_2, \ldots, x_n\}$ and then multiply them by certain parameters called weights $\in\{\theta_1, \theta_2, \ldots, \theta_n\}$, each of these products are then added together and this value is designated as "z". This sum product is passed to a non-linear function called the activation function designated as *g(z)* and the value it produces is called "a" known as the activation. In a neural network there is a concept of layers. Neurons are considered to be in the same layer when no neuron in the layer gets its input from another neuron in the same layer. The process described above is performed for all the neurons in the layer so $x_j^{(l)}$ and $a_j^{(l)}$ are the *jth* neuron and activation in layer *l*, $\theta_{ij}^{(l)}$ is the weight between the *ith* neuron in layer *l*-1 and the *jth* neuron in layer *l.* At the inception an external signal is provided to the first layer *l* = 1 (input layer) and then through the sum product and activation process this signal is transformed and propagated to the last layer *l* = L. At the last layer, $a_j^{(L)}$ is compared with what is known as the true value y_j which is produced by the primary inputs in the natural system we are trying to characterize. The difference between the value we predicted ($a_j^{(L)}$) and the true value (y_j) is the error $\delta_j^{(L)}$. This is very valuable information because it tells us exactly how much we were "off." In order to make full use of this information we need to pass it back through to the other layers; enter back-propagation. It can be shown that the subsequent deltas are calculated by $\delta_i^{(l)} = \sum_{j=1}^{n} \theta_{ij}^{(l)} \delta_j^{(l+1)} g'(z_i^{(l)})$, where "$g'$" is the derivative of *g*. Using a popular optimization method called gradient decent, the parameters are changed by updating them in the direction that brings the next prediction closer to the true value. Using a cost function that penalizes the parameters for being wrong we get the direction of that cost function's derivative to be the proper direction of change, $\frac{\partial J(\Theta)}{\partial \theta_{ij}^{(l)}} = \sum_{j=1}^{n} a_i^{(l)} \delta_j^{(l+1)}$, where $J(\Theta)$ is the cost function. All parameters are updated by $\theta_{ij}^{(l)} = \theta_{ij}^{(l)} - \alpha \frac{\partial J(\Theta)}{\partial \theta_{ij}^{(l)}}$, where α is a small number called the learning rate, until no perceivable error reduction is observed or we exceed the maximum number of iterations.

Hardware learning Implementations

In [1] Holeman et al. they explore the various circuits that could be instrumental in the implementation of different hardware learning systems including bump-circuits and configurable distance calculators. However, their analysis of computation error is the primary interest of this paper in regards to [1]. Holeman et al. discuss the obvious benefits that analog computations have in terms of power efficiency and speed but they also highlight the potential for error due to component differences and noise. However, due to the intrinsic feedback and self-organization in learning systems analog implementations become robust to these imperfections and are able to "learn" errors and optimize around them. Furthermore, their evaluation of an analog learning system compared to a digital one shows that analog systems are able to exhibit comparable accuracy to floating point computations while consuming significantly less power.

[3] explores the use of a dynamic method to perform parameter updates. In past implementations of analog learning systems, resistive networks or components were used to create static weights that could not easily be adjustable. Kawaguchi et al. explore the use of switch capacitor circuits as a way to dynamically change impedance and hence adjust weighs. Switched capacitors work by using an AC signal to move charge in a capacitor by fast switching, creating an electrically controlled resistance. Although, this is not the method of dynamic weight update employed in this paper, [3] was instrumental expounding on the necessities of having a dynamic system when performing learning tasks.

[6] was fundamental to the development of the system in this paper, Bibyk et al. propose a neuron architecture that permits dynamic parameter update and also has the benefit of being fully implantable with analog components. Using a four-quadrant vector multiplier [6] implements a neuron that can accept an arbitrary number of inputs and parameters which are represented as voltages. The proposed system is able to permit continuous-time signal processing in addition to performing vector multiplication to an incredibly high precision due to the operational amplifier operating in feedback.

[11] was particularly helpful, not for any technical consideration but for the relatively distinctive idea of separating the activation function from the neuron. Most of the analog learning systems discussed in other papers either integrate a pseudo activation function with the neuron often involving a double-inverter buffer as was done in [6] or completely ignore the need for one as in [3]. In the ANN system of this paper the activation function was implemented as a distinct sub system from the individual neurons, which has benefits in both modularity and ease of debugging.

[12] presents a fascinating implementation of a dynamic parameter update system. Using floating-gate transistors (FGT) as a memory element, Borgstrom and Bibyk propose a method of both saving and updating parameters. When a voltage pulse is provided to the gate of a floating-gate transistor, it changes the stored charge in its extra dielectric layer which increases or decreases the threshold voltage of the transistor. The gates of two FGTs are tied together and receive the same voltage pulse however their bodies are connected to different circuities that determine how the parameter update happens. The sources of both of these FGTs are connected to a current reference and the drains are connected to a current comparator which is responsible for generating the final parameter voltage. However, in this paper back propagation and parameter update is implemented digitally using a micro controller. [12] is extremely instructive for designing an extremely compact system that bypasses more of the digital overhead by taking advantage of analog options.

CHAPTER III

METHODS

In the process of designing this system many design architectures for the neural network's computational engine were evaluated before the final design was chosen. This design was chosen to be able to be able to provide reconfigurability to the system allowing it to be trained on multiple different data sets. This section will provide a detailed overview of how the architecture and design of the system was developed as well as the methods used to find and finetune parameters. Figure 1 illustrates a block diagram representation of how the system will function.

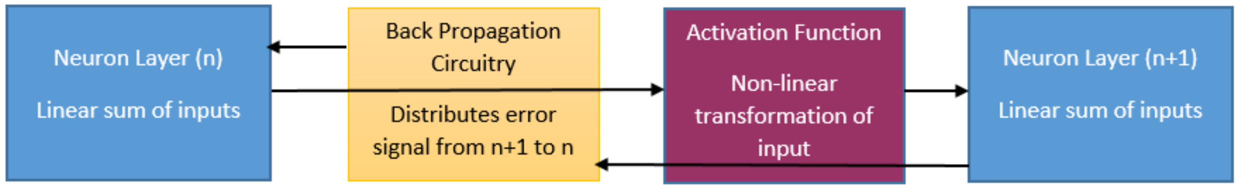

Neuron Layer (n)
Linear sum of inputs
Back Propagation Circuitry
Distributes error signal from n+1 to n
Activation Function
Non-linear transformation of input
Neuron Layer (n+1)
Linear sum of inputs

Figure 1. Block Diagram of System

The system can be divided into three main sub systems, although there are multiple instances of these subsystems occurring throughout the design. These sub systems include the Neurons, Activation function block and the Backpropagation. The neuron takes in either primary inputs or inputs from internal nodes and performs a sum-product function of the inputs and corresponding weights. The activation function block implements a non-linear transformation of the output of the neuron to produce a new value. The back-propagation algorithm relays error signals generated by the current layer to the previous layer of neurons.

Neuron Architecture

The neuron is the smallest unit cell of a neural system. For simplicity proposes, it will support up to 3 inputs and produce a single output, implemented through analog components, namely, transistors and operational amplifiers. Each perceptron will function by accepting inputs in analog voltages and finding the dot product of those inputs with weights also represented by analog voltages. Multiple perceptrons will be combined together to produce the neural network.

Architectural designs of the neuron were inspired by those of Steve Bibyk and Mohammed Ismail in [6]. A single neuron is designed by implementing a four-quadrant vector multiplier with NMOS transistors operating in the triode region and an operational amplifier. To provide inputs, which is a feature that the machine learning system will be trained on, this architecture will need 4 n-type or p-type transistors per input. Furthermore, an additional four transistors will be needed at the output of the operational amplifier to convert the source current of the input transistors to a voltage. In total one AD8607ARMZ-REEL low power operational amplifier and 16 transistors from MC14007BDR2G packages are used to realize one neuron.

Each transistor has a differential pair, so the voltage applied to the gate of a transistor is subtracted from the voltage applied to the gate of its pair. This is desirable for two reasons, also acknowledged by Steve Bibyk et al, differential signals provide a way to obtain cleaner, more accurate signals simply because common mode variations that exist in both transistors are subtracted from each other, thereby canceled. Another benefit to this architecture is that it allows the dynamic weight update functionality it needed by a non-static machine learning system. Figure 2 illustrates the schematic of a neuron cell that has three inputs. The formula governing the function of a neuron is given by

$$V_{out} = \frac{1}{V_{c1} - V_{c2}} \sum_{i=1}^{n} X_i \, W_i$$

Where n is the number inputs that the neuron can take. X_i is an input and W_i is its corresponding weight. V_{c1} and V_{c2} are calibration voltages that can be used to add an additional weighting to specific neurons.

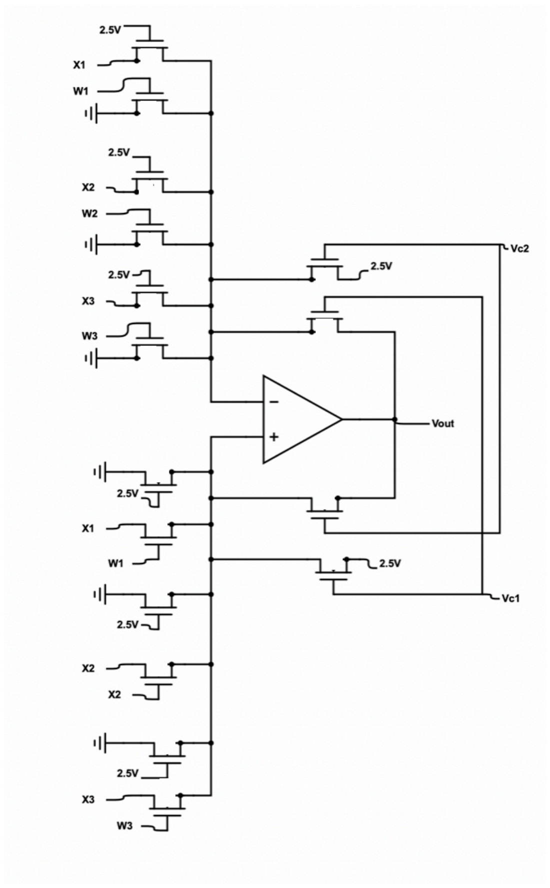


Figure 2. Schematic of Neuron

In this application each neuron will be neutral so V_{c1} and V_{c2} have values 5V and 4V respectively, this is implemented so no neuron can have a bias that isn't created by the data under analysis. The, possibility of dynamically adjusting the values of V_{c1} and V_{c2} was shortly explored as an additional mechanism that the data analysis process can control, or as a way to factor out terms from the weights (W values) to allow a larger range. However, for the sake of system simplicity only fundamental functions of a neural network are implemented and tested in the system. Refer to [6] for a more in-depth analysis of this neuron architecture. Two constraints (equation 1 and 2) that limit the range of operation of the system will be highlighted as it becomes important in analysis of how operational values such as V_{c1}, V_{c2}, and max (x_i) were chosen.

$$V_{out} \leq \min[V_{c1} - V_{th}, V_{c2} - V_{th}] \dots (1)$$

$$|X_i| \leq |V_{th}| \dots (2)$$

Consequently, this architecture limits the maximum output voltage we can hope to reliably produce which explains why the largest possible values of V_{c1} and V_{c2} were chosen while still maintaining a 5 watt of power consumption, as well as neutral neuron weighting. In most transistors the thermal voltage (V_{th}), an intrinsic parameter of the device, is in the range of 380 mV to 500 mV [7,8]. Thus, this gives us an approximate output voltage range of 0 to 3.5V where the transistors do not become reverse biased. Furthermore, the highest input voltage we can reliably hope to deliver to the neuron from any one input transistor must be less than 0.38V.

To test the range of voltages that can be used for the weight values, I observed the results of performing a DC sweep of weights, on the simulated circuit, from -2V to 2V while leaving other voltage sources as constant. As can be observed from Figure 3, we experience very linear behavior in this range of voltages, with a minuscule error of -0.2mV due to the offset of the operational amplifier. Through further testing to optimize system parameters, the optimal weights where determined to take a value in the range from -1.2V to 1.2V, and the inputs from 0V to 0.3V. Although, one weight can swing from -2V to 2V with no a problem and one input from 0 to 1V, all three weights and inputs are unable to do so at the same time without causing significant distortion at the output. These optimal weight values were also verified through the following calculations:

$$V_{out_{max}} = 3.5V,\ X_{i_{max}} = 0.3V$$

$$V_{out_{max}} \leq 3W_{i_{max}}X_{i_{max}}$$

$$W_{i_{max}} \leq \ 1.296V$$

The 2.5V source, like the 4V source, is created with a TLV757 linear regulator IC to bias various transistors. The weights in the system will be operating relative to a voltage of 2.5V reference in order to realize “negative” voltages. i.e. the actual weight values are shifted up by 2.5V. Each neuron in the overall system relays its output voltage to an activation function subsystem which converts the voltage by a nonlinear transformation. A maximum input value of 0.3V volts was chosen in order not to reverse bias the transistors. Further probabilistic analysis in the backpropagation subsystem report will show that these are a sufficient range of values.

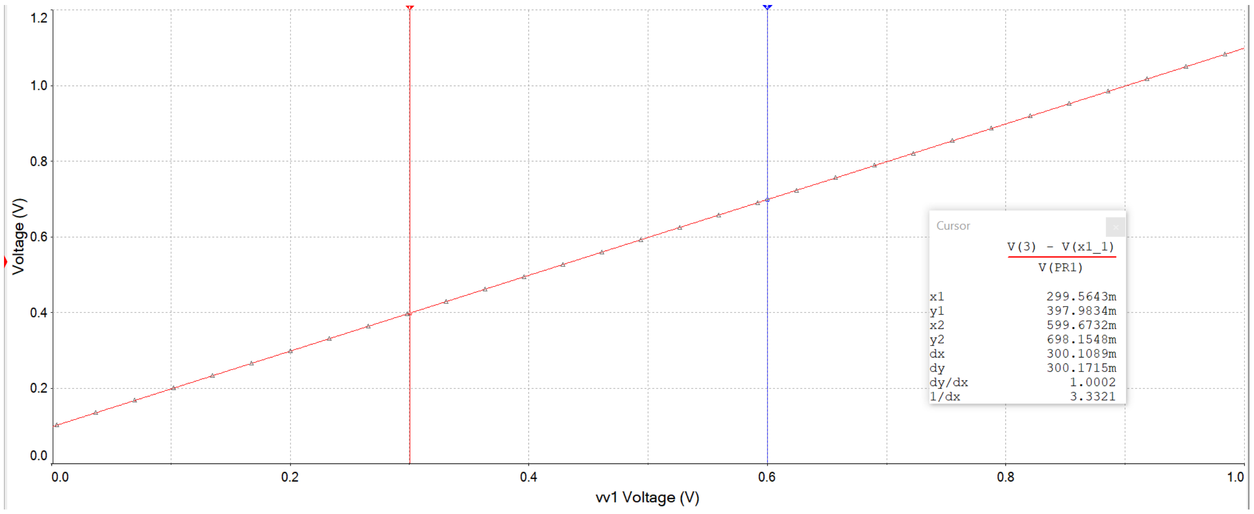


Figure 3: DC sweep of Neuron Input 1 from 0V to 1V

The neuron in the first layer will receive its inputs as DC voltage values from a PWM-DC converter. The PWM signals will be provided by an Arduino Microcontroller and has a resolution of 10bits. The input of every other neuron that is not in the first layer will receive its inputs from the neuron in the previous layer. The weights of the neuron are controlled by the Arduino PWM signals which will be adjusted through the back-propagation algorithm. Problems experienced with this system is the inability to provide negative input voltages due to the architecture of the neuron. This problem will be solved by normalizing (setting the mean to 0 and the standard deviation to 1) the input values and shifting them above zero. Statistically, this way no information is lost because it is a linear transform. Furthermore, by normalizing our inputs, with range of values covered in enough standard deviations, we are able to represent a large proportion of inputs. This is guaranteed by Chebyshev's inequality. Which states that the probability to observe a value in any distribution that is away from its mean value by k standard deviations (σ) is less than $\frac{1}{k^2}$.

Activation Function System Architecture

The activation function system will take the output of a perceptron and apply a non-linear function f(x), to map the input to a new value. This subsystem will be able to support switching between different functions such as the Rectified Linear Unit (ReLU), and Sigmoid activation functions. Inputs to each activation function block is the value produced by each neuron and the output is the nonlinear transformation computed by the system. A 74HC4052D 4:1 Multiplexer is used to select the desired activation function that is to be used. Two digital signals provided by the Arduino pins 22 and 24 indicate which one of the two activation functions to use.

Sigmoid

The sigmoid function is a pseudo logistical function governed by equation (3):

$$f(x) = \frac{1}{1+e^{-x}} \dots (3)$$

This function was produced with two CMOS chipsets, implemented as a buffer that was scaled to swing to a maximum value of 0.3V (instead of 1 as in Figure 4) when the input (the output of the neuron) reaches about 1.1V. This choice was taken to make sure that the voltage produced by the activation function subsystem is useable by the next neuron in the chain which can only accept a maximum input of 0.3 V. The value 1.1V was calculated by assuming that all three sets of weights and input voltages for a neuron take their maximum value of 1.2V and 0.3V respectively. In this event the maximum output a single neuron can produce is 1.08V. The sigmoid function produces output voltages relative to a 2.5V reference, this way we are able to express "negative" voltages. The sigmoid was simulated and tested for its range of expected inputs and performs as expected, the system is able to take a range of inputs and then map them to a point on the sigmoidal curve, producing an output. Figure 5 and 6 illustrates the circuitry and its performance respectively. As can be observed from Figure 6 we experience a maximum output voltage of 0.3 V for an input of 1.08V and an output voltage of 0V for -1.08V.

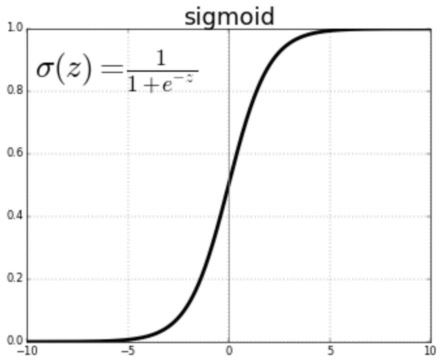


Figure 4: Sigmoid

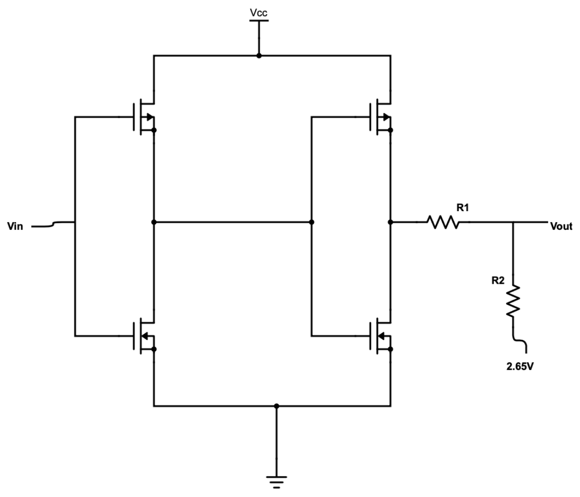


Figure 5: Circuit schematic for Sigmoid function

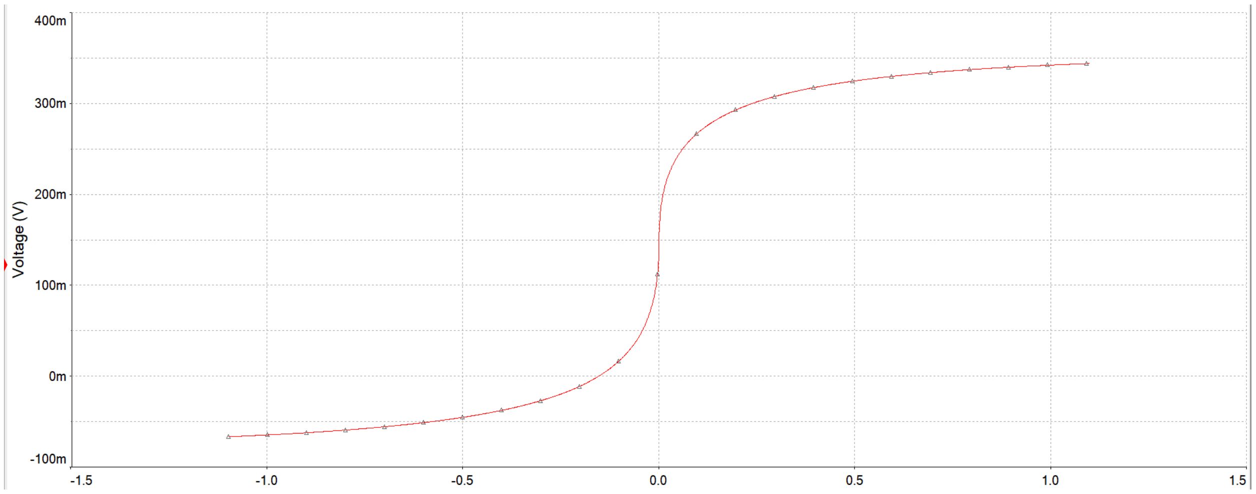


Figure 6: Transfer characteristics of buffer circuit

Rectified Linear Unit (ReLU)

The ReLU activation is a function that takes in an input and maps it to its self if the input following equation (4).

$$f(x) = \begin{cases} x, & if \;\; x \geq 0 \\ 0, & if \; x < 0 \end{cases} \dots (4)$$

This is a choice activation function for a lot of machine learning applications because of its simplicity and intuitive mapping scheme. The ReLU system is implemented by a precision half wave rectifier using an AD8607ARMZ-REEL low power operational amplifier and a diode in what is commonly referred to as a “super-diode” as can be observed from the schematic shown in Figure 7. When the input (Vin) at the positive terminal of the operational amplifier is positive the negative terminal sees the same value because of a virtual short between them. The output is then equal to the input because the negative terminal and the output are connected. However, when there is a negative input at the positive terminal, the diode becomes revered biased and does not let current back into the diode. With nowhere for current to go (the high input resistance of an op-amp also resists the flow of current), there is now a virtual short between ground and the output, setting the output to 0V. This configuration does come with a limitation: we begin to experience saturation for high input voltages (~4.6V). However, the inputs were designed to never reach that threshold. A voltage divider is also used to regulate the output magnitude of the ReLU function. Using the same rationale as that used in the sigmoid system, for a maximum input voltage a 1.1V we are expecting no more than a 0.3V output. The resistors in the voltage divider were sized appropriately to achieve this purpose. Also, similar to the sigmoid function all voltages used by the system is relative to a 2.5V reference.

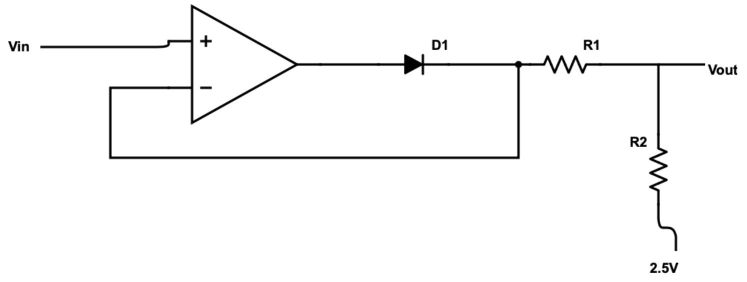


Figure 7: Circuit schematic for ReLU function

As can be observed from Figure 8, the system acts according to (4). However, as mentioned previously the scaling is adjusted by roughly a 3:1 ratio so at a maximum possible input of 1.1V we see 0.3V at the output.

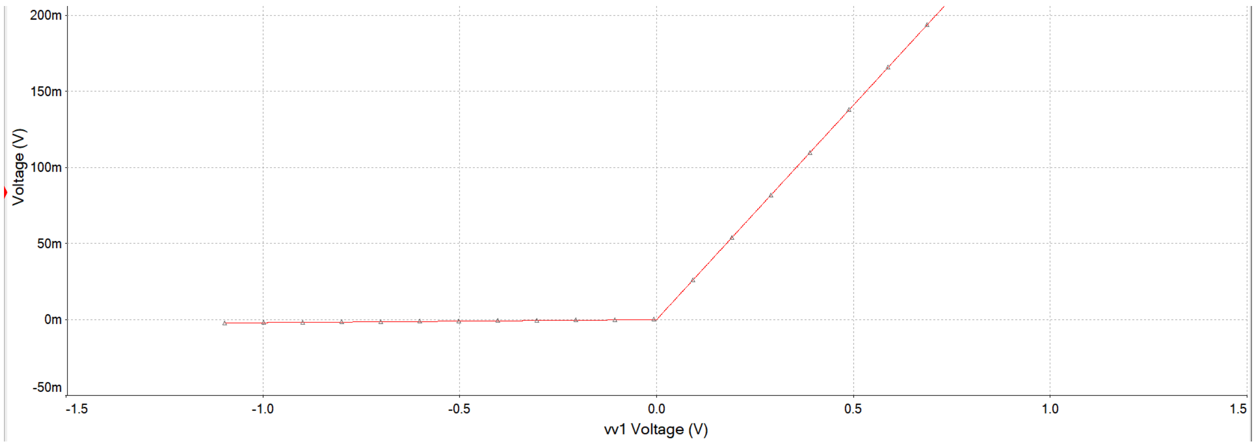


Figure 8: Transfer characteristics of ReLU circuit

Back Propagation System Architecture

The back-propagation system will be essential to the overall performance of the device. Back propagation allows the error detected at the output layer of the Neural Network to be sent backward through each layer so weights can be adjusted to provide a better prediction in the next learning round. This system will be implemented with an Arduino microcontroller which will also act as the host computer. Figure 9 below illustrates a simplified flow chart of how the back-propagation algorithm works. For a more detailed analysis refer to [9,10]. This process iteratively compiles errors between the predicted value and the true value. And then adjusts the weights of each neuron by subtracting a delta value (which could be either negative or positive) when this process is done may times (epochs) for all the inputs, we converge to an optimal set of parameters that are able to accurately predict the output values when given a set of inputs.

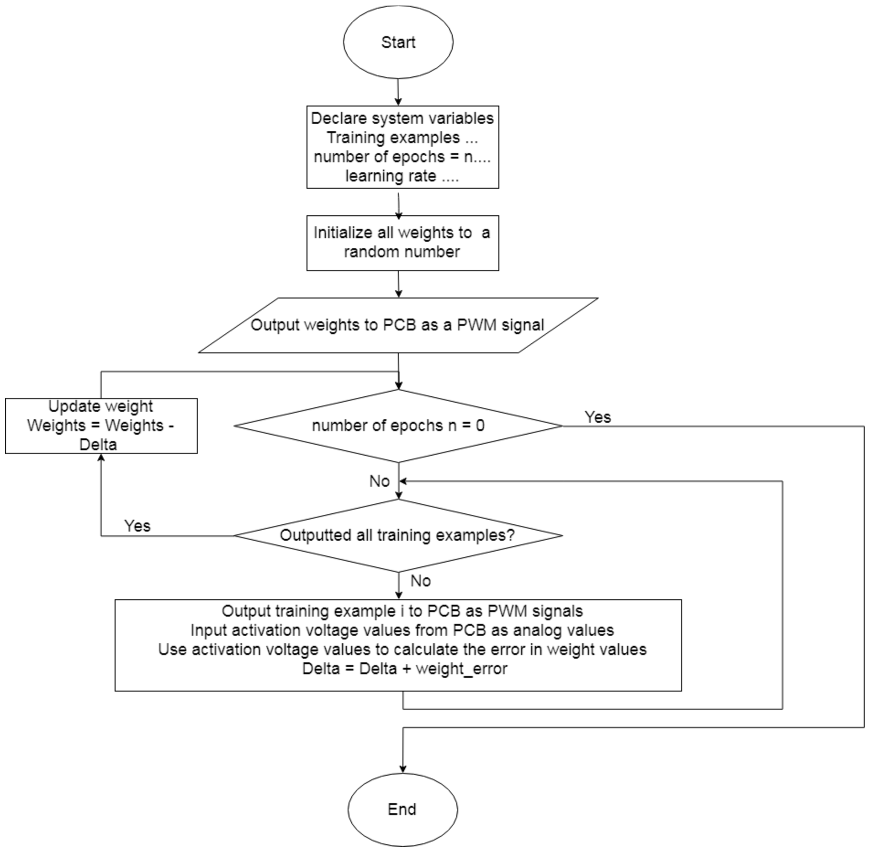

Start
Declare system variables
Training examples ...
number of epochs = n....
learning rate ....
Initialize all weights to a random number
Output weights to PCB as a PWM signal
Update weight
Weights = Weights - Delta
number of epochs n = 0
Yes
No
Yes
Outputted all training examples?
No
Output training example i to PCB as PWM signals
Input activation voltage values from PCB as analog values
Use activation voltage values to calculate the error in weight values
Delta = Delta + weight_error
End

Figure 9. Flow Chart of Back-Propagation Algorithm

This system is handled by the Arduino microcontroller and is written in C code. The Arduino provides 3 primary inputs and a bias term of "1" and the weights to the forward propagation system (neuron and activation function). After the input is provided by the Arduino in a 10bit PWM format, a LTC2645CMS PWM-DC converter converts this to a DC value, and feeds it into the first neuron which provides an output which gets propagated into the activation function system which then produces an output called the activation. Each of the three activations (output of the activation systems) named a1, a2, and a3 are inputs to both the next neuron as well as to the Arduino through analog input pins. These activation values are used in the calculation of the delta (error), which is aggregated for all the input training examples, and used to adjust each of the nine weights. After delta is determined the weights are adjusted and outputted to begin the process again for a predefined number of epochs.

Due to the input value constraints created by the neuron architecture, PWM signals, and PWM-DC converter. We are forced to resort to a probabilistic guarantee that we can support a large enough range of values. We are able to account for up to 10 standard deviations, by setting 30mV to represent one standard deviation (or unit), on either side of the mean of input values. By Chebyshev inequality, statistically, there is less than a 1% chance that we will not be able to accurately represent an input. With a 10bit PWM signal we can express voltages as small as 5mV which gives us a resolution 6 times that of our unit. Furthermore, with maximum weight values of ±1.2 volts and a unit of 30mV we can represent weights that are the equivalent of 40 units in magnitude. A weighting this large in a learning system is able to accurately convey the importance of an input. More so, the back-propagation algorithm will be using regularization. This will limit the values the weights can achieve further showing that a rage of ±40 units is an ample range.

CHAPTER IV

RESULTS

Initial simulations of the three subsystems were conducted using Multisim Circuit Simulator. As can be observed from Figures 3, 6 and 8 we experience very accurate results with the most deviation being in the neuron with an error of one-five thousandth of a volt. After a prototype was implemented on a Printed Circuit Board (PCB) further tests were done on the functionality of the whole system with the deferent subsystems integrated together.

The input layer is able to take in 3 inputs and produce an activation value that is passed to the next layer which subsequently passes its own activation to the outermost layer. Each layer also receives a bias value of 1. Given the extremely simple architecture of the system, having three layers and one neuron per layer, it was given a very simple learning task as a proof of concept. For regression training dummy test data of three inputs was created. Three normal random variables $X_1 = N(\mu = 3, \sigma = 1)$, $X_2 = N(\mu = 5, \sigma = 1)$, and $X_3 = N(\mu = 7, \sigma = 1)$ was produced by a python script and the underlying function that the system will be trained to recognize is Y = $2(X_1 + X_2) + X_3$. For classification training the same inputs were used with classification criteria:

$$Y = \begin{cases} 0 & if \quad X_1 + X_2 + X_3 \geq 15 \\ 1 & Otherwise \end{cases}$$

Only two classes are being predicted because with one output a simple neuron architecture can only reliably indicate a binary prediction. Figure 10 illustrates the plot of Y versus the different features where Figures a, b, and c are the inputs X_1, X_2 *and* X_3 respectively. And Figure 11 shows the class distribution for the classification task.

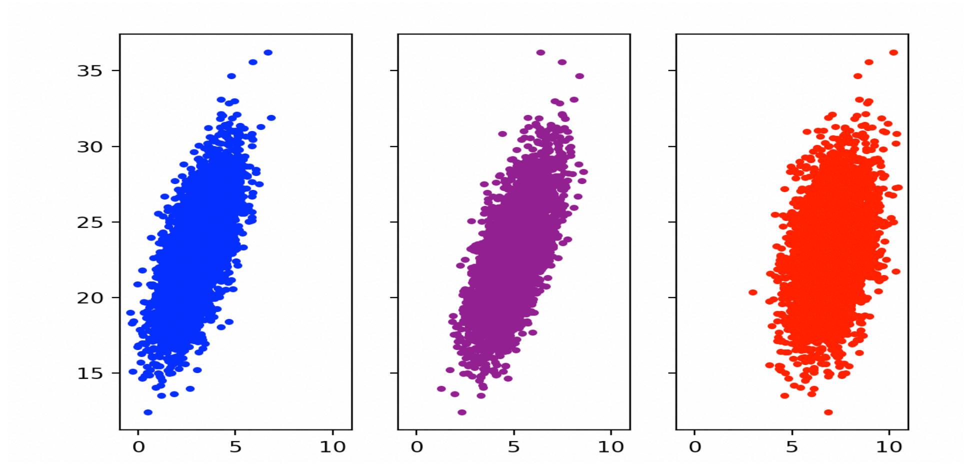


Figure 10. Plot of Features X_1, X_2 *and* X_3 against Y

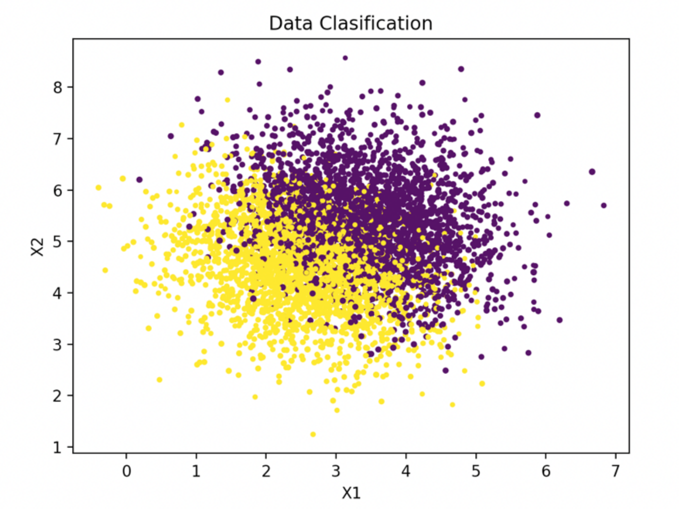


Figure 11. Class distribution of for classification task

Fifty thousand examples split 80-20 were generated to train and test the neural network system. And the following sections illustrated the performance of the overall system according to different metrics.

Training Speed

The system was trained for 1 epoch for each training set. Figure 12 shows the function of training time for different training sizes. Both regression and classification exhibited similar responses. As can be observed from the graph the training time increases linearly as a function of number training examples with a factor of 0.022, allowing the system to be trained on 45 inputs in 1 second. The analog computational hardware only takes roughly 318μ seconds to perform the forward propagation and the remaining time per training example is a function of the microcontroller, host computer's clock speed and the baud transfer speed between them.

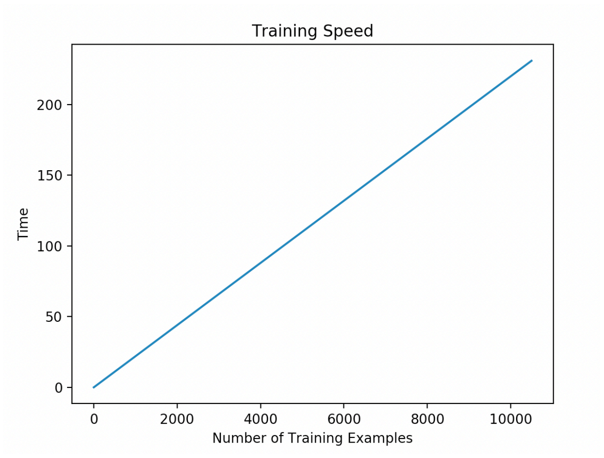


Figure 12. Plot of training time against number of test examples

Prediction Speed

The system was used to make predictions on data that was held out during the training phase. Figure 13 illustrates the graph of prediction time versus number of examples. Both regression and classification exhibited similar responses. As in the case of prediction the graph increases linearly as a function of the number testing examples with a factor of 0.018, allowing the system to make 55 predictions in 1 second. Similar to the case of training, the analog computational only takes roughly 318μ seconds to perform the forward propagation.

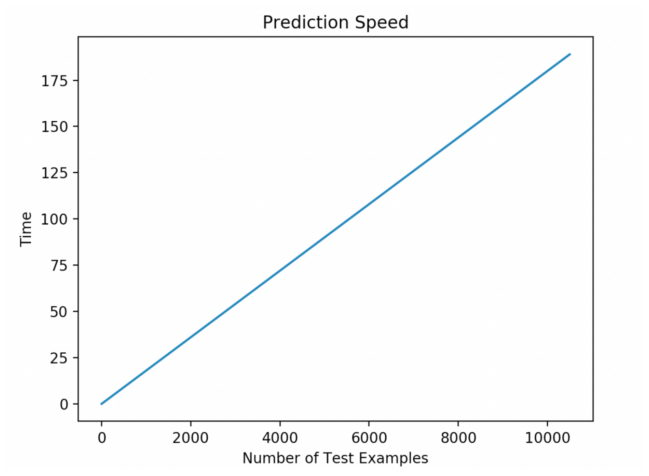


Figure 13. Plot of prediction time against number of test examples

Peak Power Consumption

During the training period the power consumption of the system was being measured and the peak power consumption was determined to be 2.435W. A similar phenomenon was observed during testing/prediction, the system experienced a peak power consumption of 2.422W.

Prediction Accuracy

Regression

For the regression task the ReLU activation function was used because of its continuous nature. Figure 14 shows the error as a function of the number of epochs the system was being trained on. At roughly 120 epochs we stop seeing any reduction in the Mean Absolute Error (MAE). The lowest MAE was achieved was after training the system was 3.734. This poor performance is attributed to the type of transistors used to implement the neuron. The prototype was implemented with different transistors than was used in the simulation. As a result of this, the transistor is unable to stay in the linear region causing the operational amplifier to saturate for voltages above ~0.058V. However, the Figure 14 does show that the system is able to somewhat learn and reduce its error.

$$MAE = \frac{1}{N}\sum_{i=1}^{N} | X_{true} - X_{pred} |$$

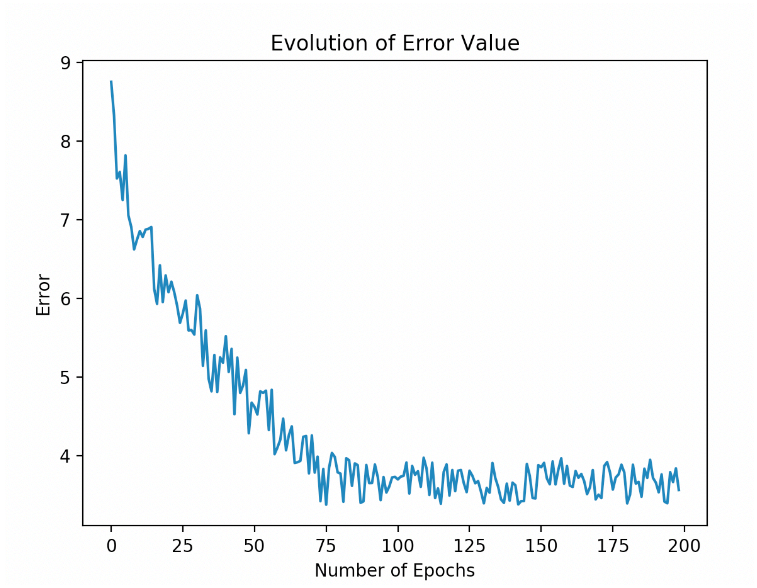


Figure 14. MAE against number of Training Epochs

Classification

In classification the sigmoid activation function was the natural choice because of its close resemblance to the logistic function. Figure 15 depict the receiver-operator characteristics curve (ROC curve) of the classifier's performance. This curve is sufficient to illustrate the performance of a classifier on two classes [13,14]. The horizontal axis is the false positive rate (specificity) and the vertical axis is the true positive rate (recall). A classifier that produces an area under the ROC curve (auROC) of 0.5 is considered a random classifier and one that produced an area of 1 is considered a perfect classifier. The system attains a testing auROC of 0.692. As explained in the regression section this performance can be significantly improved by using transistors that possess the desired characteristics.

$$Recall = \frac{TP}{TP + FN}$$

$$Specificity = \frac{FP}{FP + TN}$$

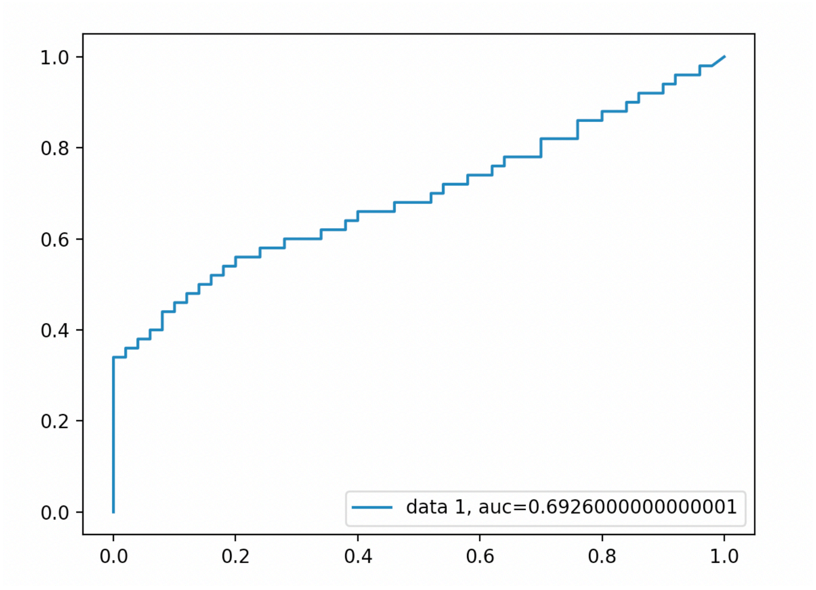
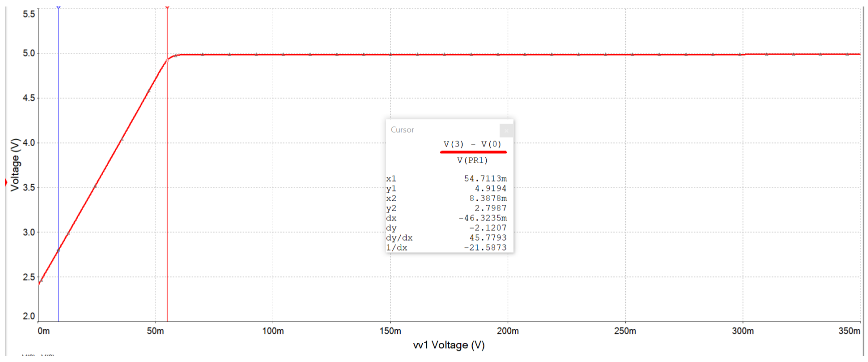


Figure 15. Plot of recall against specificity for classification

Figure 16 shows the simulation result of a neuron circuit using the transistors assembled on the board. The behavior in the simulation matches what is observed in. When the total sup-product of the voltage entering the operational amplifier exceeds 0.058V the output of the nueron saturates and rails to its 5V supply. Furthermore, the linear region before the operational amplifier rails is not usable because it produces a slope that is roughly 46 times what is expected.



```
            V(3) - V(0)
              V(PR1)

x1              54.7113m
y1                4.9194
x2               8.3878m
y2                2.7987
dx             -46.3235m
dy               -2.1207
dy/dx            45.7793
1/dx            -21.5873
```

Figure 16. Simulation results using the transistors assembled on the PCB

CHAPTER V

CONCLUSION

With the advent of big data and the desire to produce analytical results from such data, we have run in to a bottleneck of resources. Firstly, the software and inherently serial machine learning techniques cannot be scaled sustainably to the required complexity of the future. Secondly, the transmission of data over the internet to the cloud for processing possess certain limitations when it comes to privacy and speed. In this paper we have proposed a possible system that will be able to perform machine learning vis a vis a neural network. The use of analog components in implementing the basic building blocks of a neural network, neuron and activation function, allows this design to scale linearly only in the number of layers. Furthermore, the modular design of this system allows it to be expanded and modified very easily.

The system is able to learn patterns to a precision of 0.69 in roughly 0.022 seconds per training example and able to make predictions in 0.018 seconds. However, the underwhelming performance of the system can be greatly attributed to the simplicity in the ANN architecture which was not sufficiently able to learn the patterns in the data. Also, the use of transistors with the wrong characteristics greatly contributed to the sub-par performance.

Furthermore, there are many future directions to take this project. The most logical step is developing an integrated circuit. Due to the highly modular design of this system it's components can be drastically miniaturized, allowing a great expansion of the ANN architecture to a much more complex one. Using current fabrication techniques, we can control the characteristics of the transistors to match the proper specifications, optimizing performance.

Also, by developing an IC, we will be able to take advantage of much shorter wirelengths and smaller components that are designed to specification. These optimizations will reduce the overhead to increase both the speed and power efficiency of the system.

Although there is the inherent potential for error in analog computations the system is able to self-organize allowing it to be robust to most errors, thus preventing performance loss. The success of this system will be able to show that machine learning capabilities cannot only become faster but can also be more power efficient. The increased computational speed that we can gain from edge computing techniques will allow us to make technologies like autonomous vehicles, optimized traffic management, and other real-time artificial intelligence a possibility.

REFERENCES

[1]J. Holleman, I. Arel, S. Young, and J. Lu "Analog Inference Circuits for Deep Learning" pp. 1

[2]J. Lu, T. Yang, M.S. Jahan, and J. Holleman, "Nano-power tunable bump circuits using wide input-range pseudo-differential transconductor," Electronic Letters, Vol.50, No. 13, pp. 921-923

[3]M. Kawaguchi, and M. Umeno "Analog Neural Circuit with Switched Capacitor and Design of Deep LearningModel" pp. 322

[4]M. Kawaguchi, T. Jimbo, and M. Umeno, "Motion Detecting Artificial Retina Model by Two-Dimensional Multi-Layered Analog Electronic Circuits" IEICE Transactions, E86-A-2, pp. 387-395

[5]M. Kawaguchi, T. Jimbo, and M. Umeno, "Analog VLSI Layout Design of Advanced Image Processing for Artificial Vision Model, " IEEE International Symposium on Industrial Electronics, ISIE2005 Proceeding, vol.3, pp.1239-1244

[6] Steve Bibyk, and Mohammed Ismail. "Issues in Analog VLSI and MOS Techniques for Neural Computing"

[7] Sudhakar, J. G. et al. "Exhaustive analysis & behavior of nanometer MOSFET for threshold voltage variations." *2015 3rd International Conference on Signal Processing, Communication and Networking (ICSCN)* (2015): 1-6.

[8] Narendra, Siva G. "Effect of MOSFET Threshold Voltage Variation on High-Performance Circuits." (2002).

[9] Cilimkovic, Mirza. "Neural Networks and Back Propagation Algorithm." (2010)

[10] Petrini, Mircea. "Improvements to The Backpropagation Algorithm." www.upet.ro/annals/economics/pdf/2012/part4/Petrini-2.pdf. (2012)

[11] Shrinath, Abhishek K. “Analog VLSI implementation of Neural Network Architecture.” (2013)

[12]Brogstrom, Tom and Bibyk, Steve. “A Neural network Integrated Circuit Utilizing Programable Threshold Voltage Devices.” (1989)

[13] Bradley, Andrew. “The Use of the Area Under the ROC Curve in the Evaluation of Machine Learning Algorithms.” Pattern Recognition. 30. (1996): 1145 - 1159.

[14] Park, Seong Ho et al. “Receiver operating characteristic (ROC) curve: practical review for radiologists” Korean journal of radiology vol. 5,1 (2004): 11-8.