Self-trained Multi-layer Analog Real-Time Artificial Neural Network Circuits

D. Rancour$^1$ and H. Michel$^2$

$^1$Electrical and Computer Engineering, UMass Dartmouth, North Dartmouth, Massachusetts, USA
$^2$Electrical and Electronics Engineering Science, University of Johannesburg, Johannesburg, South Africa

Abstract - A “Self-trained Multi-layer Analog Real-Time” (SMART) ANN architecture solves the pattern recognition problem for numbers 0-9 in about 20 µs. The Single hidden-Layer Feed Forward Neural Network is realized as an Extreme Learning Machine and thus allows random selection of the hidden-layer synapse values, leaving just the output-layer synapse values to be determined. SMART training circuits find the output-layer synapse weight values. The analog electronic ANNs and training circuitry provide real-time training as well as real-time data processing. Feedback loops make continuous adjustments of the ANN synapse weights while in training mode. Training for all weights occurs in parallel, at the response speed of the analog electronics regardless of the number of neurons.

Keywords: Self-trained ANN, analog ANN, real-time ANN training

1 Introduction

The time required to train Artificial Neural Networks (ANN) can range from seconds to days, depending on the scale of the neural network and the training method. Moreover, ANNs are typically realized as software emulations run on computers that execute their instructions serially, thereby losing the advantages of ANNs’ inherent parallelism. In contrast, an analog circuit implementation of ANNs can provide both fast training and real-time operation. Our “Self-trained Multi-layer Analog Real-Time” (SMART) ANN architecture solves the pattern recognition problem for numbers 0-9, in about 20 µs$^1$.

Each of the numbers 0-9 is defined by a 7 x 5 grid of black & white pixels, therefore the first layer of the perceptron has 35 inputs. The output-layer must have 10 neurons to identify the 10 digits. The number of neurons in the hidden-layer is also ten, as shown in Figure 1 below.

This architecture constitutes a Single hidden-Layer Feed Forward Neural Network (SLFF-NN) and can thus be realized as an Extreme Learning Machine (ELM) [1]. The ELM method allows random selection of the 35 x 10 = 350 hidden-layer synapse values, leaving just 10 x 10 = 100 output-layer synapse values to be determined. Here the 350 fixed hidden-layer synapse values are uniformly distributed between ±10 V, and SMART training circuits find the remaining 100 output-layer synapse weight values in real time.

The remainder of this paper is organized as follows. Section 2 presents related work. Section 3 shows the analog electronic circuits used to realize the SMART architecture. Section 4 gives SPICE simulation results. Section 5 presents a summary.

2 Related work

Intel Corp marketed an analog ANN chip (Intel 80170NX, a.k.a. ETANN) in the early 1990’s. ETANN gave fast data processing, but training was done off chip and therefore was still slow [2]. Cancelo and Hansen developed supporting hardware and software to facilitate training ETANN, but still only implemented standard training algorithms and so did not achieve real-time on-chip training [3].

IBM Corp together with General Vision Inc. sold the first commercial digital ANN chip, the ZISC (Zero Instruction Set Computer), from 1993 to 2001. Again, training was done off-chip. Moreover “… early neural networks were outperformed by other algorithms, such as support vector machines in image processing and Gaussian models in speech recognition …” [4] and interest in the technology waned. After 2001 IBM ceased commercial manufacture of the ZISC chip. However, advances in hardware and progress on “Deep Learning” methods revived interest in ANNs [4].

Apart from IBM, General Vision continued developing its ZISC technology, to produce their NeuroMem chip [5]. The NeuroMem chip is built in SRAM and it excels at parallel pattern matching. One of its features is to be able to match one pattern versus many patterns in a constant time. It can be 100,000, a million or more patterns. The task takes 30 clock cycles (2.5 µs) regardless of the number. Recognition and learning latencies are constant, on the order of micro-seconds per pattern and independent of the number of neurons in use.

$^1$ Transient simulations in Multisim®.
ZISC technology is near real-time, but it is a digital system requiring a fixed number (30) of clock cycles to perform each recognition and/or learning task. General Vision has licensed its IP for use in 1) the Intel Curie/Quark SE processor, and 2) facial recognition systems realized in FPGAs [6].

In 2014, under the auspices of DARPA’s “SyNAPSE” program, IBM announced its “TrueNorth” neuromorphic chip [7]. TrueNorth is a digital 28 nm technology with 1 million programmable neurons and 256 million programmable synapses complete with its own custom programming language. Training is done off-chip and therefore is not real-time.

Qualcomm acquired neural network startups in 2013, and recently launched its Qualcomm® Zeroth™ platform [8]. Qualcomm Zeroth technology began as a hardware NPU (neural processing unit) and was expanded into a software design kit that uses Qualcomm’s Snapdragon processors. Details of the technology are proprietary but it is digital and therefore, at best, it has near real-time training.

Alternatives to the classic “Back Propagation” (BP) training algorithm have been shown to provide speed benefits. “Particle Swarm Optimization” (PSO) [9], [10] has greater parallelism than BP and so is faster, but PSO is still an iterative algorithm and therefore cannot achieve real-time training even when implemented in hardware.

The “Extreme Learning Machine” (ELM) [1], [11], [12] algorithm is a non-iterative fixed-time training method for SLFF-NNs. ELM can achieve, at best, near real-time learning speed and is only applicable to SLFF-NNs i.e. ANNs with one hidden layer.

3 SMART circuits

We generate each of the 10 input pixel patterns in Multisim using arrays of $5 \times 7 = 35$ dc voltage sources. Figure 2 below shows the particular dc voltage source array used to represent the numeral “2”. Each pixel is set to a value of either $+0.5 \, \text{V}$ or $-0.5 \, \text{V}$. The $+0.5 \, \text{V}$ pixels are circled in Figure 2, and it can be seen that they are positioned to roughly map out the numeral “2”. Numerals 0-9 were realized in a similar manner.

The aggregation function for each hidden-layer neuron consists of 35 interconnected AD633AN$^2$ analog multiplier chips. The basic functional block diagram for the AD633 is shown in Figure 3 below. The chip’s transfer function is $W = \frac{(x_1-x_2)(y_1-y_2)}{10 \, \text{V}} + Z$. 

\[ W = \frac{(x_1-x_2)(y_1-y_2)}{10 \, \text{V}} + Z. \]

\[ W = \frac{(x_1-x_2)(y_1-y_2)}{10 \, \text{V}} + Z. \]

\[ W = \frac{(x_1-x_2)(y_1-y_2)}{10 \, \text{V}} + Z. \]
The $X_2$ and $Y_2$ inputs are set to $0 \, V$, the $X_1$ inputs are driven by input pixel voltages and the $Y_1$ inputs are driven by synapse weight voltages (uniformly distributed between $\pm 10 \, V$). The multipliers are daisy chained by connecting successive outputs $W$ to inputs $Z$ to realize the aggregate. pn diode clipping circuits, as shown in Figure 4 below, provide the hidden-layer activation functions.

![Figure 2: Array of 5×7=35 dc voltage sources realizing the pixel pattern for the numeral “2”. Circled sources are +0.5 V, all other sources are −0.5 V.](image)

AD633AN multiplier chips also realize the aggregation function for each output-layer neuron, however, because training circuits must find the 100 output-layer synapse weight values these multipliers are not daisy chained; this avoids daisy chain ripple delay during training. Here inputs $Z$ are set to $0 \, V$ and outputs $W$ connect to op amp adders to execute the sums in parallel rather than in sequence via daisy chains. Figure 5 shows an output-layer neuron aggregation circuit using an AD826AN$^3$ inverting Op amp adder.

![Figure 3: Functional block diagram for ANALOG DEVICES AD633 four-quadrant analog multiplier chip.](image)

Figure 6 presents the training circuitry for a single output-layer neuron, and includes the electronics of Figure 5 as abstract multiplier and adder symbols. There are ten training circuits in total, one for each output-layer neuron, i.e. one for each of the numerals 0-9. Signals “Weight1”, “Weight2”... “Weight10” are common to all ten output-layer neurons, whereas each output-layer neuron receives a unique set of “Activation” signals from the hidden layer, namely the signals generated by that neuron’s corresponding input numeral. For example, neuron eight gets “Activation” signals generated by input numeral “8”, etc.

The output-layer signal “Aggr” drives “Op amp1” to generate the signal “Weight1” which is captured by the track-and-hold circuit. The track-and-hold circuit feeds “Weight1” back to the output-layer neuron to form the product “Weight1 × “Activation1”. The polarity of the feedback loop therefore depends on the sign of the hidden-layer neuron signal “Activation1”; so analog comparator “Comp1” sets the input connections of “Op amp1” through switch “S1” to guarantee a negative feedback configuration. Although the sign of the hidden-layer signal “Activation1” can be either “+” or “−”, once its value is randomly determined it remains fixed, i.e. it does not change during training.

![Figure 4: Activation function circuit and its voltage transfer characteristic curve.](image)

$^3$ Only three of the ten multiplier blocks are explicitly shown.
Figure 5: Output-layer neuron aggregation circuit.

Figure 6: Training circuit.
The output-layer signal “Aggr” also drives analog comparator “Comp2” which controls switches “S2” and “S3” to set the training circuit to either its “track” or “hold” mode. Switch “S2” moreover keeps “Op amp1” from going open loop while in “hold” mode.

The user sets switch “S4” to achieve the desired training outcome: either “pattern recognized” or “pattern ignored”. With “S4” in its “up” state, as shown in Figure 6, “Comp2” puts “S2” and “S3” in track mode for “Aggr” > 0. For “Aggr” < 0 “Comp2” puts “S2” and “S3” in hold mode and the feedback loop opens thereby locking down “Weight1”. This corresponds to “pattern ignored”. “Weight1” stays fixed as long as “Aggr” < 0, but will readjust if “Aggr” wanders above zero during training.

With “S4” in its “down” state “Comp2” puts “S2” and “S3” in track mode for “Aggr” < 0. For “Aggr” > 0 “Comp2” puts “S2” and “S3” in hold mode and the feedback loop opens thereby locking down “Weight1”. This corresponds to “pattern recognized”. “Weight1” stays fixed as long as “Aggr” > 0, but will readjust if “Aggr” wanders below zero during training.

4 Simulation results

The ten training circuits in essence provide ten equations in the ten unknown weights. Each training circuit has feedback control of one specific weight value. All ten output neurons receive the same ten weights. Each output neuron receives its own unique set of “Activation” signals corresponding to the specific input numeral that the neuron is to “recognize” or “ignore”. For example, to recognize an “8”, the user would put output neuron 8 in “recognize” mode while putting the remaining nine neurons in “ignore” mode. Training ends when all output-layer synapse weights lock, giving the specific settings for the neuron’s ten weights that result in “Aggr” > 0 for input “8” and “Aggr” < 0 for the other nine input numerals. Figure 7 below illustrates this specific case.

The figure shows ten “Aggr” output voltage transients. During the first 6 μs all ten outputs have “Aggr” < 0, thus nine of the ten output neurons have their training circuits in hold mode while output neuron 8’s training circuit is in track mode. So, during the first 6 μs, only one of the ten weights is adjusted by feedback and the other nine weights are locked down. However all ten output neurons receive the same ten weights, so all ten “Aggr” outputs change in response to the single adjusted weight. Some of the “Aggr” outputs swing up and others down because each output neuron has its own unique set of “Activation” signals from the hidden layer, some with positive values and others with negative values.

By the 6.5 μs mark, feedback has driven the “Aggr” output of neuron 8 above the 0 V threshold, thereby switching its training circuit to hold mode. Meanwhile five other “Aggr” outputs have also risen above the 0 V threshold, putting their training circuits in track mode thus enabling feedback which forces them back below 0 V. After about 20 μs, all ten training circuits are in hold mode and all weights are locked, signifying recognition of the numeral “8”. Similar results were obtained for recognition of digits 0-7, and 9.

4.1 Feedback weight assignments

Each output-layer neuron produces an “Aggr” signal that drives an op amp to generate one of the ten weight signals. There are $10! = 3,628,800$ different possible ways to configure the feedback weight assignments. The weight values may fail to lock down if arbitrary weight assignments are made in disregard of certain “selection rules”. This is due to the contention that can occur whenever two or more training

![Figure 7: Output voltage transients for ten aggregation function circuits configured to recognize “8” and ignore digits 0-7, and 9.](image)

Figure 6 shows “S2” and “S3” in their “hold” state.
circuits are simultaneously put into track mode. The training circuits can vie for control, unless their weight assignments are made according to the selection rules. The selection rules take into account both the signs and magnitudes of the “Activation” signals. Figure 8 shows an example of how the ten the “Aggr” signals can oscillate if the weight assignments violate the selection rules.

Figure 8: Failure of output voltage transients to converge.

5 Summary

ANNs are loosely modeled after biological systems, mimicking the neurons and synapses of the brain. However, ANNs typically do not mimic the complex feedback mechanisms prevalent in physiology. Here we have pushed the simulation of ANNs somewhat closer to the actual situation for living systems, where the players are constantly being adjusted via intricate feedback processes. We have demonstrated a SMART ANN architecture which solves the pattern recognition problem for numbers 0-9, in about 20 µs, using an assemblage of standard electronic components found in Multisim®. This demonstrates proof-of-concept. Our proposed SMART architecture is truly real-time for both ANN function and training; if fabricated as an ASIC it will perform both tasks in nanoseconds.

6 References


