Chapter 2

Background
To facilitate a better understanding of the work embodied in the thesis, an overview of the necessary background information is presented in this chapter. It includes a discussion on the Hopfield network including comments on feasibility and limitations of neural network solutions based on the Hopfield approach. Thereafter, another recurrent architecture – the NOSYNN, is explained. An application for which the NOSYNN has already been applied, viz. graph coloring, is also discussed and it is shown that the performance of the NOSYNN based graph coloring network can be improved by a simple modification in the existing network [137]. The chosen problems on which the newly proposed (voltage-, mixed-, and current-mode) hardware implementations of the NOSYNN have been tested are then explained. Thereafter, an overview of available neural network based solutions for the chosen problems is presented.

2.1 Hopfield Neural Network

In his seminal paper of 1982, John Hopfield described a new way of modeling a system of neurons capable of performing ‘computational’ tasks [74]. The Hopfield neural network emerged, initially as a means of exhibiting a content-addressable memory (CAM). A general CAM must be capable of retrieving a complete item from the system’s memory when presented with only sufficient partial information. Hopfield showed that his model was not only capable of correctly yielding an entire memory from any portion of sufficient size, but also included some capacity for generalization, familiarity recognition, categorization, error correction, and time-sequence retention. The Hopfield network, as described in [74, 73, 155, 75], comprises of a fully interconnected system of $n$ computational elements or neurons. The strength of the connection, or weight, between neuron $i$ and neuron $j$ is determined by $W_{ij}$, which may be positive or negative depending on whether the neurons act in an excitatory or inhibitory manner. The internal state of each neuron $u_i$ is equivalent to the weighted sum of the external states of all connecting neurons. The external
state of neuron $i$ is given by $v_i$, with $-1 \leq v_i \leq 1$. An external input, $i_i$, to each neuron $i$ is also incorporated.

The relationship between the internal state of a neuron and its output level in this continuous Hopfield network is determined by an activation function $g_i(u_i)$. Commonly, this activation function is given by

$$v_i = g_i(u_i) = \tanh \left( \frac{u_i}{T} \right)$$

(2.1)

where $T$ is a parameter used to control the gain (or slope) of the activation function. A typical plot of the activation function $g_i$ is presented in Figure 2.1. In the biological neural system, on which the Hopfield network is based, $u_i$ lags behind the instantaneous outputs, $v_j$, of the other neurons because of the input capacitance, $C_i$, of the cell membrane, the trans-membrane resistance $R_i$, and the finite impedance $R_{ij}(= W_{ij}^{-1})$ between the output $v_j$ and the cell body of neuron $i$. Thus, the following resistance-capacitance differential equation determines the rate of change of $u_i$, and hence the time evolution of the continuous Hopfield network:

$$C_i \frac{du_i}{dt} = \sum_j W_{ij}v_j - \frac{u_i}{R_i} + i_i$$

(2.2)
2.1 Hopfield Neural Network

Figure 2.2: An electronic circuit representation of the continuous Hopfield neural network

\[ u_i = g_i^{-1}(v_i) \]  \hspace{1cm} (2.3)

The set of equations (2.2, 2.3) can also be represented by a resistively connected network of electronic amplifiers shown in Figure 2.2 with the \( i \)-th neuron of the network of Figure 2.2 shown in Figure 2.3. The synapse (or weight) between two neurons is now defined by a conductance \( W_{ij} \), which connects one of the two outputs of amplifier \( j \) to the input of amplifier \( i \). The connection is made with a resistor of value \( R_{ij} (= W_{ij}^{-1}) \). Figure 2.3 also includes an input resistance, \( R_{pi} \), for amplifier \( i \). This \( R_{pi} \) is present implicitly in equation (2.2) as being included in \( R_i \) which is a parallel combination of \( R_{pi} \) and \( R_{ij} \):

\[ \frac{1}{R_i} = \frac{1}{R_{pi}} + \sum_j \frac{1}{R_{ij}} \]  \hspace{1cm} (2.4)
Node equation for node ‘A’ gives the equation of motion of the $i$-th neuron of Figure 2.3 as

$$C_{pi} \frac{du_i}{dt} = \frac{v_1}{R_{i1}} + \frac{v_2}{R_{i2}} + \ldots + \frac{v_n}{R_{in}} - u_i \left[ \frac{1}{R_i} \right] + i_i$$  \hspace{1cm} (2.5)

which can be written as

$$C_{pi} \frac{du_i}{dt} = \sum_j \left[ \frac{v_j}{R_{ij}} \right] - u_i \left[ \frac{1}{R_i} \right] + i_i$$  \hspace{1cm} (2.6)

It can be seen that the (2.6) is the same as (2.2) with the weights being implemented by resistances. For simplicity, each neuron/amplifier is assumed to be identical, so that

$$R_i = R; C_i = C; g_i = g;$$  \hspace{1cm} (2.7)

Dividing (2.2) by $C$ and redefining $W_{ij}/C$ and $i_i/C$ to be $W_{ij}$ and $i_i$, respectively, we arrive at the normalized equations of motion:

$$\frac{du_i}{dt} = \sum_j W_{ij}v_j - \frac{u_i}{\tau} + i_i$$  \hspace{1cm} (2.8)

$$\tau = RC$$  \hspace{1cm} (2.9)

$\tau$ is the value of the time constant of the amplifiers, and without loss of generality can be assigned a value of unity, provided the time step of the discrete

\[ \text{Figure 2.3: } i\text{-th neuron of Hopfield neural network} \]
time simulation of (2.8) is considerably smaller than unity. Although this ‘neural’ computational network has been described in terms of an electronic circuit, it has been shown [75] that biological models with action potentials and excitatory and inhibitory synapses can compute in a similar fashion to this electrical hardware.

To ascertain the stability of the system described by equations (2.2, 2.3), Hopfield employed the concept of a so called ‘generalized computational energy function,’ $E$, which is essentially a Lyapunov function associated with the network which guarantees convergence to stable states. For the continuous Hopfield network, the $i$-th neuron of which is shown in Figure 2.3, the gradient of the energy function is related to the evolution of the neuronal state as

$$\frac{\partial E}{\partial V_i} = -C_{pi} \frac{du_i}{dt}; \text{ for all } i$$

Using (2.6) and (2.10), the energy function corresponding to the network of Figure 3.4 can be written as

$$E = -\frac{1}{2} \sum_i \sum_j W_{ij} v_i v_j - \sum_i i_i v_i + \sum_i \frac{1}{R_i} \int_0^{v_i} g_i^{-1}(v) \, dv$$

Provided the matrix of weights $W$ is symmetric (although Vidyasagar [159] has shown that convergence is still possible under some asymmetric conditions), the time derivative of $E$ is:

$$\frac{dE}{dt} = \sum_{i=1}^{N} \frac{\partial E}{\partial v_i} \frac{dv_i}{dt} = \sum_{i=1}^{N} \frac{\partial E}{\partial v_i} \frac{du_i}{dt}$$

Using (2.10) in (2.12) we get

$$\frac{dE}{dt} = -\sum_{i=1}^{N} C_i \left(\frac{du_i}{dt}\right)^2 \frac{dv_i}{du_i}$$

The transfer characteristics of the opamp used in Figure 2.3 implements the activation function of the neuron. With $u_i$ being the internal state at the non-inverting terminal, it is monotonically increasing as shown in Figure 2.1, and therefore,

$$\frac{dv_i}{du_i} \geq 0$$
thereby resulting in
\[ \frac{dE}{dt} \leq 0 \] (2.15)

with the equality being valid for
\[ \frac{du_i}{dt} = 0 \] (2.16)

Together with the boundedness of \( E \), (2.15) shows that under control of the differential equation (2.6), \( E \) decreases and converges to a minimum, at which it stays.

Hopfield, along with Tank, realized that the analog nature of the neurons and the parallel processing of the updating procedure could be combined to create a rapid and powerful solution technique. They presented several applications of the Hopfield neural network including Content Addressable Memory (CAM), solution of Travelling Salesman Problem (TSP), analog-to-digital conversion, signal decision and linear programming. The underlying idea in most of these applications was the fact that solution to specific optimization problems could be obtained by selecting weights and external inputs that appropriately represent the function to be minimized and the desired states of the problem. In other words, the network energy function is made equivalent to the objective function of the optimization problem that needs to be minimized, while the constraints of the problem are included in the energy function as penalty terms. Once a suitable energy function has been chosen, the network parameters (weights and inputs) can be inferred by comparison with the standard energy function given by (2.11). The weights of the continuous Hopfield network, \( W_{ij} \), are the coefficients of the quadratic terms \( v_i v_j \), and the external inputs, \( i_i \), are the coefficients of the linear terms \( v_i \) in the chosen energy function. The network can then be initialized by setting the activity level \( v_i \) of each neuron to a small random perturbation. From its initialized state, asynchronous updating of the network will then allow a minimum energy state to be attained.
However, these stable states may not necessarily correspond to *feasible* or *good* solutions of the optimization problem, and this is one of the major pitfalls of the Hopfield neural network. Because the energy function comprises several terms (each of which is competing to be minimized), there are many local minima, and a tradeoff exists between which terms will be minimized. An infeasible solution to the problem will arise when at least one of the constraint penalty terms is non-zero. If this occurs, the objective function term is generally quite small, because it has been minimized to the detriment of the constraint terms, thus the solution is “good” but not feasible. Alternatively, all constraints may be satisfied, but a local minimum may be encountered that does not globally minimize the objective function, in which case the solution is feasible but not “good.” To overcome such a situation, a penalty parameter can be increased to force its associated term to be minimized, but this generally has the unwanted effect of causing other terms to be increased. The solution to this trade-off problem is to find the optimal values of the penalty parameters that balance the terms of the energy function and ensure that each term is minimized. Only then will the constraint terms be zero (a feasible solution), and the objective function be also minimized (a “good” solution).

Further, due to certain omissions in Hopfield and Tank’s paper [73] relating to termination criteria, simulation procedure for differential equations, etc. it has been quite difficult to reproduce the results for TSP. Wilson and Pawley were the first to report this discrepancy and suggested many modifications in the original Hopfield neural network [174]. Over time, two distinct approaches to improve the original Hopfield neural network have been developed. The first approach consists of rewriting the energy function to eliminate the trade-offs between valid and good solutions. The second technique is to accept the Hopfield energy function as such while searching for ways to optimally select the penalty parameters. A systematic method for selection of parameters based on the analysis of dynamic stability of valid solutions has been developed by Kamgar-Parsi and Kamgar-Parsi [85].
2.1 Hopfield Neural Network

Efforts to obtain better results by modification of energy function have proved to be more fruitful. Significant contributions in this regard include the valid subspace approach of Aiyer et al. [4] and the subsequent work by Gee [55]. Other approaches of modifying the energy function, specific to the TSP problem, have also been reported [42, 127]. Deterministic methods like the “divide and conquer” technique of Foo [49], “rock and roll” perturbation method of Lo [106], and the use of alternative neuron models within the Hopfield network such as the winner-take-all neurons [6], to improve the quality of solution have also been presented.

Although the fundamental worth of Hopfield network is beyond contention, the never ending quest to improve solution quality has resulted in new models of neuron dynamics being investigated. Chaotic neural networks [26] and hybridization of neural networks with meta-heuristics such as simulated annealing and genetic algorithms [10] have been offered as alternatives to the Hopfield neural network.

Due to the limitations mentioned above as well as the requirement of a large number of neurons and interconnection weights, attempts to apply Hopfield networks in obtaining working hardware for practical applications have not proved to be encouraging. For instance, to solve a N–city TSP problem, a Hopfield network requires N^2 neurons and O(N^4) weights. This translates to the requirement of 100 neurons and 10000 weights for a 10–city TSP problem thereby clearly indicating that the Hopfield network is not amenable to a hardware realization. Alternative neural network architectures were therefore explored, some of which are mentioned above. Another neural architecture which employs non-linear feedback, as opposed to linear feedback in the Hopfield network, has been proposed as an alternative to the Hopfield network [136]. It has been shown that the architecture facilitates more efficient hardware implementations for combinatorial problems as compared to the Hopfield network [136]. The details of the non-linear feedback neural network are discussed next.
2.2 Nonlinear Synapse Neural Network

A recurrent neural network architecture, termed as Nonlinear Synapse Neural Network (NOSYNN), has been proposed as an alternative to the Hopfield network [136]. In a manner similar to the Hopfield network, the NOSYNN is comprised of a single layer of neurons and a fully interconnected feedback structure. The most important difference between the two is in the nature of feedback. While the Hopfield network employs linear synapses, the NOSYNN uses nonlinear synapses to which input signals are fed back from the neuron outputs. The nonlinearity in the synapse is implemented by using comparators in the circuit realization of the network. The provision of a direct feedback from a neuron to itself is also present as opposed to the original Hopfield network in which self-interactions in the neurons were not allowed.

Figure 2.4 presents the schematic of the NOSYNN and the corresponding electronic circuit implementation of NOSYNN is presented in Figure 2.5. The internal state and output of the $i$-th neuron are denoted by $u_i$ and $V_i$ respectively. $S_{ij}$ represent the synapses giving feedback from the $j$-th neuron to the $i$-th neuron. The output of $S_{ij}$ is denoted by $x_{ij}$, which is a non-linear function of $V_i$ and $V_j$, and is given by

$$x_{ij} = V_m \tanh \beta (V_j - V_i)$$

(2.17)

where $\pm V_m$ are the maximum output levels of the comparator and $\beta$ is the gain of the comparator. For a high gain comparator, the transfer characteristics are practically indistinguishable from a signum function, as shown in Figure 2.6. Individual resistors are used to assign different weights to the comparator outputs. To introduce a weight $W_{ij}$ in the feedback connection from the $j$-th neuron to the $i$-th neuron, a resistance $R_{ij}$ is employed where

$$R_{ij} = \frac{1}{W_{ij}}$$

(2.18)

Another noteworthy feature of the NOSYNN is the possibility of obtaining negative weights by simply interchanging the comparator inputs. This is
2.2 Nonlinear Synapse Neural Network

Figure 2.4: Schematic of the NOSYNN architecture [136]
Figure 2.5: Electronic circuit implementation of the NOSYNN
possible due to the fact that the comparator transfer characteristics are given by the \( \tanh(.) \) function which is an odd function. This eliminates the need of generating inverted amplifier outputs for implementing negative weights, as is required in the Hopfield network.

The equation of motion of the \( i \)-th neuron may be obtained by applying KCL at the node \( u_i \), which is the internal state of the \( i \)-th neuron. From Figure 2.5, the following differential equation is obtained.

\[
C_{pi} \frac{du_i}{dt} + \frac{u_i}{r_{pi}} = I_i + \frac{V_i - u_i}{R_{ii}} + \sum_{j=1, j \neq i}^{n} \frac{x_{ij} - u_i}{R_{ij}} \tag{2.19}
\]

Using (2.19) and (2.17), we have

\[
C_{pi} \frac{du_i}{dt} + \frac{u_i}{r_{pi}} = I_i + \frac{V_i - u_i}{R_{ii}} + \sum_{j=1, j \neq i}^{n} \frac{V_m \tanh(\beta(V_j - V_i)) - u_i}{R_{ij}} \tag{2.20}
\]

which can be further simplified to

\[
C_{pi} \frac{du_i}{dt} = I_i + \frac{V_i - u_i}{R_{ii}} + \sum_{j=1, j \neq i}^{n} \frac{V_m \tanh(\beta(V_j - V_i))}{R_{ij}} \tag{2.21}
\]
where
\[
\frac{1}{R_i} = \frac{1}{r_{pi}} + \frac{1}{R_{ii}} + \sum_{j=1, j\neq i}^{n} \frac{1}{R_{ij}} \tag{2.22}
\]

Therefore, \(R_i\) is equivalent to the parallel combination of all the resistances connected at the input of the neuron \(i.e.\) to the node at internal state \(u_i\). At the steady state, the internal and output states of the neuron may be obtained by putting the left-hand side of (2.21) equal to 0, and are given as

\[
u_i = I_i + \frac{V_i}{R_{ii}} + \sum_{j=1, j\neq i}^{n} \frac{V_m \tanh(\beta(V_j - V_i))}{R_{ij}}\tag{2.23}
\]

\[
V_i = -\tanh(\lambda u_i) \tag{2.24}
\]

where \(\lambda\) is the gain of the operational amplifier used to implement the activation function of the neuron. It is to be noted that both \(R_i\) and \(C_i\) have been normalized to unity without any loss of generality.

The NOSYNN may be associated with the following Lyapunov function [136]:

\[
E = \sum_{i=1}^{n} \left[ I_i V_i + \frac{V_i^2}{2R_{ii}} - \frac{1}{2} \sum_{j=1, j\neq i}^{n} \frac{V_m \ln \left[ \cosh \beta (V_j - V_i) \right]}{\beta} - \frac{1}{R_i} \int_{0}^{V_i} u_i dV_i \right] \tag{2.25}
\]

It has been assumed that the weights are symmetric \(i.e.\) \(W_{ij} = W_{ji}, (i,j = 1,2,\ldots,n)\). Due to the inclusion of nonlinearity in the feedback path (implemented by using comparators), the Lyapunov function for the NOSYNN, as given in (2.25) is considerably different from the corresponding function for the standard Hopfield network, given in (2.11). While the Hopfield network is associated with a Lyapunov function containing quadratic terms, the NOSYNN is characterized by transcendental terms in the Lyapunov function.

The fact that \(E\), given in (2.25) is indeed a valid Lyapunov function for the NOSYNN can be proved as follows. The \(i\)-th component of the gradient
of $E$ is given by

$$\frac{\partial E}{\partial V_i} = I_i + \frac{V_i}{R_{ii}} - u_i \frac{V_m \tanh \beta (V_j - V_i)}{R_{ij}}$$  \hspace{1cm} (2.26)$$

Comparing (2.21) and (2.26), with $C_{pi}$ normalized to unity, we get

$$\frac{\partial E}{\partial V_i} = \frac{du_i}{dt}$$  \hspace{1cm} (2.27)$$

The total derivative of $E$ with respect to time is given by

$$\frac{dE}{dt} = \sum_i \frac{\partial E}{\partial V_i} \frac{dV_i}{du_i} \frac{du_i}{dt}$$  \hspace{1cm} (2.28)$$

From (2.27) and (2.28), we have

$$\frac{dE}{dt} = \sum_i \frac{dV_i}{du_i} \left( \frac{du_i}{dt} \right)^2$$  \hspace{1cm} (2.29)$$

From (2.24), we have

$$\frac{dV_i}{du_i} \leq 0$$  \hspace{1cm} (2.30)$$

Using (2.30) in (2.29), we get

$$\frac{dE}{dt} \leq 0$$  \hspace{1cm} (2.31)$$

which proves that $E$ is non-increasing with time, thereby fulfilling one (of two) condition for a valid Lyapunov function. The second requirement of $E$ being bounded from below can be justified by considering the fact that since the amplifier outputs saturate, the neuron states are bounded from below as well as from above. Therefore, $E$, which is a function of the neuron outputs and is bounded for bounded arguments, will also be bounded.

The Lyapunov function associated with feedback neural networks, like the Hopfield network and the NOSYNN, has been extensively used to solve hard optimization problems. Such Lyapunov function based networks are guaranteed to settle at a stable output corresponding to one of its minima. If a particular network is so designed that the minima of its associated Lyapunov function (also referred to as the ‘energy’ function) correspond to solution states
of a given optimization problem, then the network can be viewed as a dynamical computing system for solving the specific problem. The applicability of NOSYNN in solving combinatorial optimization problems like graph coloring and ranking, by proper selection of the ‘energy’ function, has been demonstrated [137, 81]. It is to be mentioned however that although the network of [137] yielded better coloring results alongwith reduced circuit complexity, the performance of the network can be further improved by incorporating a slight alteration in the circuit. The details of such a modified NOSYNN based graph coloring network are presented next.

2.2.1 Graph Coloring using Modified Voltage-mode NOSYNN

The graph colouring problem requires assigning values, labels or colours to the nodes of a given graph with the constraint that adjacent nodes be coloured distinctly, i.e. assigned different labels, values, or colours; while requiring the minimum number (referred to as the chromatic number of the graph) of colours. A proper colouring requires the constraints to be strictly satisfied, and therefore none of the adjacent nodes should get the same colour. The GCP and its variants have applications in many important tasks such as timetabling or event scheduling [82], register and processor allocation in digital computers [54], frequency or channel assignment in mobile communication [47] and layer assignment in VLSI design [98]. The recreational puzzle Sudoku can be visualized as a ‘constrained-to-nine-labels’ colouring on a specific graph with 81 vertices.

The NOSYNN when applied to solve GCP yielded promising results [136]. Figure 2.7 presents the $i$-th neuron of the NOSYNN based voltage-mode neural circuit for graph colouring. The output of the $i$-th neuron, $V_i$, corresponds to a voltage label (colour) assigned to the $i$-th node. Selective connections are provided from the output of the neurons to the comparators connected in the feedback path. The voltage feedback from a node, which is connected (adjacent) to the $i$-th node, is allowed as an input of the comparator connected in the feedback path. For non-adjacent nodes, no feedback signal arrives at
2.2 Nonlinear Synapse Neural Network

Figure 2.7: i-th neuron in the NOSYNN based circuit for graph colouring. (Feedback to comparators is from all neurons corresponding to the nodes which are adjacent to the i-th node: \( j = 1, 2, \ldots, n; j \neq i \))

![Diagram of neuron circuit](image)

the input of the i-th neuron, thereby meaning that no comparators would be present for such cases. The value of \( R_C \) was chosen to be 1 KΩ and the self-feedback resistance for each neuron was calculated as

\[
R_{ii} = \frac{R_C}{D}
\]  

where \( D \) is the maximum degree amongst all nodes in the graph to be coloured [136].

Although the network, the i-th neuron of which is presented in Figure 2.7, outperforms other existing hardware solutions for GCP [137], a minor modification in the circuit can bring about further improvements in the colouring performance of the network. The comparators used in the actual realization of the graph coloring NOSYNN based neural network [137] have bipolar transfer characteristics as shown in Figure 2.6. The range of allowable voltages for the output values of various nodes is \( \pm V_m \). The network then assigns voltages
(traditionally called ‘colours’) to different nodes in the graph depending upon their adjacencies. This assignment is done by forcing the neuron to take only one out of a finite number of allowed discrete voltage levels from $[-V_m, V_m]$ for the steady state outputs \[136\].

However, it was observed that if the allowable range of the discrete values at the output of neurons is restricted to $[0, V_m]$ or $[-V_m, 0]$, the network can be forced to assign colours from a smaller set of available colours, thereby having the effect of reduction in the number of colours. This can be achieved by employing a unipolar comparator instead of a bipolar one as used in Figure 2.7. The transfer characteristics of a voltage-mode unipolar comparator are presented in Figure 2.8. As can be seen, two possibilities exist: (a) the outputs being 0 or $V_m$ (b) the outputs being $-V_m$ or 0. One possible circuit realization for obtaining the transfer characteristics of Figure 2.8 is presented in Figure 7.3, which shows a diode used in conjunction with the voltage-mode comparator yielding the characteristics as given in Figure 2.6.

Figure 2.10 shows the $i$-th neuron of modified voltage-mode network for graph colouring. In the proposed circuit, output voltages of different neurons represent the colors of different nodes. $C_{pi}$ and $r_{pi}$ denote the internal capac-
2.2 Nonlinear Synapse Neural Network

Figure 2.9: Obtaining the unipolar characteristics of Figure 2.8 by using a bipolar comparator and a diode; (a) Circuit for obtaining the transfer characteristics of Figure 2.8(a); (b) Circuit for obtaining the transfer characteristics of Figure 2.8(b).

...ittance and resistance of the $i$-th neuron respectively, $u_i$ is the internal state and $R_{ii}$ is the self-feedback resistance of $i$-th neuron. The output of other neurons $V_j; (j = 1, 2, \ldots, n)$ are connected to the input of $i$-th neuron through unipolar comparators.

The self-feedback resistance $R_{ii}$ can be calculated using (2.32). Next, in order to ascertain values of the resistances connected in the synaptic paths, we define a constant $g_{ij}$ such that

$$g_{ij} = \begin{cases} 1; & \text{$i$-th node connected to $j$-th node} \\ 0; & \text{otherwise} \end{cases}$$ (2.33)

Using (2.33), the resistance in the $j$-th synapse of the $i$-th neuron is given by

$$R_{ij} = \frac{R_c}{g_{ij}}$$ (2.34)

From Figure 2.10, $x_{ij}$ can be written as

$$x_{ij} = \frac{V_m}{2} \left[ \tanh \beta (V_j - V_i) - 1 \right]$$ (2.35)

where $\beta$ is the open-loop gain of the comparator (practically very high) and $-V_m$ is the saturation voltage level of the comparator output.
2.2 Nonlinear Synapse Neural Network

Figure 2.10: $i$-th neuron of the modified voltage-mode graph colouring neural network based on NOSYNN

Node equation for node ‘A’ gives the equation of motion of the $i$-th neuron in the state space as

$$C_i \frac{du_i}{dt} = \sum_{j=1}^{N} \frac{x_{ij}}{R_{ij}} + \frac{V_i}{R_{ii}} - \frac{u_i}{R_i}$$  \hspace{1cm} (2.36)

where

$$\frac{1}{R_i} = \sum_{j=1}^{N} \frac{1}{R_{ij}} + \frac{1}{R_{ii}} + \frac{1}{r_i}$$  \hspace{1cm} (2.37)

The NOSYNN-based graph coloring network of Figure 2.7, which employs bipolar voltage-mode comparators, is associated with the following energy function [136]:

$$E = \frac{1}{2} \sum_{i=1}^{N} \frac{V_i^2}{2R_{ii}} - \frac{V_m}{4\beta R_c} \sum_{i=1}^{N} \sum_{j=1}^{N} g_{ij} \ln \cosh (\beta (V_j - V_i))$$
Table 2.1: Hardware implementation and PSPICE simulation test results for the proposed network

<table>
<thead>
<tr>
<th>S. No.</th>
<th>Test Graph</th>
<th>Hardware Results</th>
<th>Simulation Results</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Voltages of different nodes with node no.</td>
<td>No. of colors</td>
<td>Frequency of occurrence</td>
</tr>
<tr>
<td>1</td>
<td>$0 \quad 2V/3 \quad 0 \quad 2V/3 \quad 0 \quad 2V/3 \quad 0 \quad 2V/3$</td>
<td>2</td>
<td>25/25</td>
</tr>
<tr>
<td></td>
<td>$0 \quad V/3 \quad 0 \quad V/3 \quad 0 \quad V/3 \quad 0 \quad V/3$</td>
<td>3</td>
<td>22/25</td>
</tr>
<tr>
<td>2</td>
<td>$0 \quad V \quad 0 \quad V \quad 0 \quad V \quad 0 \quad V$</td>
<td>3</td>
<td>25/25</td>
</tr>
<tr>
<td></td>
<td>$0 \quad V \quad 0 \quad V \quad 0 \quad V \quad 0 \quad V \quad 0 \quad V \quad 0 \quad V \quad 0 \quad V \quad 0 \quad V$</td>
<td>2</td>
<td>25/25</td>
</tr>
</tbody>
</table>

\[ E = \frac{1}{2} \sum_{i=1}^{N} \frac{V_i^2}{2R_{ii}} - \frac{V_m}{4\beta R_c} \sum_{i=1}^{N} \sum_{j=1}^{N} g_{ij} \ln \cosh (\beta (V_j - V_i)) \]

\[ - \sum_{i=1}^{N} \frac{1}{R_i} \int_{0}^{V_i} u_i dV \]  

(2.38)

Using (2.38), and considering that the comparator outputs are now given by (2.35) instead of (2.17), the energy function corresponding to the modified NOSYNN-based voltage-mode network for graph coloring of Figure 2.10 can be written as

\[ E = \frac{1}{2} \sum_{i=1}^{N} \frac{V_i^2}{2R_{ii}} - \frac{V_m}{4\beta R_c} \sum_{i=1}^{N} \sum_{j=1}^{N} g_{ij} \ln \cosh (\beta (V_j - V_i)) \]

\[ - \sum_{i=1}^{N} \frac{1}{R_i} \int_{0}^{V_i} u_i dV \]  

(2.39)
2.2 Nonlinear Synapse Neural Network

Table 2.2: Performance comparison of the proposed network with the NOSYNN-based graph coloring network of Figure 2.7

<table>
<thead>
<tr>
<th>S. No.</th>
<th>Test Graph</th>
<th>Simulation Results</th>
<th>Simulation Results</th>
<th>Chromatic Number</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>NOSYNN-based Neural Network for Graph Coloring</td>
<td>Modified NOSYNN-based Neural Network for Graph Coloring</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Minimum number of colors</td>
<td>Average number of colors in 10 trials</td>
<td>Minimum number of colors</td>
<td>Average number of colors in 10 trials</td>
</tr>
<tr>
<td>1</td>
<td><img src="image1" alt="Graph" /></td>
<td>4</td>
<td>4.5</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td><img src="image2" alt="Graph" /></td>
<td>4</td>
<td>4.4</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td><img src="image3" alt="Graph" /></td>
<td>7</td>
<td>7.8</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td><img src="image4" alt="Graph" /></td>
<td>7</td>
<td>8.2</td>
<td>5</td>
</tr>
</tbody>
</table>

In order to prove the validity of the energy function of (2.39), the time derivative of $E$ can be obtained as

$$
\frac{dE}{dt} = \sum_{i=1}^{N} \frac{\partial E}{\partial V_i} \frac{dv_i}{dt} = \sum_{i=1}^{N} \frac{\partial E}{\partial V_i} \frac{dV_i}{dt} \frac{du_i}{dt}
$$

(2.40)

Also, for the NOSYNN-based dynamical systems, it has been shown that [136]

$$
\frac{dE}{dV_i} = C_p \frac{du_i}{dt}
$$

(2.41)

Using (2.10) in (2.12) we get

$$
\frac{dE}{dt} = \sum_{i=1}^{N} C_i \left( \frac{du_i}{dt} \right)^2 \frac{dV_i}{du_i}
$$

(2.42)
2.2 Nonlinear Synapse Neural Network

Figure 2.11: Transfer characteristics of the opamp used to realize the neuron in Figure 2.7

The transfer characteristics of the opamp used in Figure 2.7 implements the activation function of the neuron. With $u_i$ being the internal state at the inverting terminal, a typical plot of the characteristics is shown in Figure 2.11 from where it is evident that the plot is monotonically decreasing, and therefore,

$$\frac{dV_i}{du_i} \leq 0$$  \hspace{1cm} (2.43)

thereby resulting in

$$\frac{dE}{dt} \leq 0$$  \hspace{1cm} (2.44)

with the equality being valid for

$$\frac{du_i}{dt} = 0$$  \hspace{1cm} (2.45)

Equation (2.44) shows that the energy function can never increase with time which is one of the conditions for a valid energy function. The second criterion *viz.* the energy function must have a lower bound is also satisfied for the circuit of Figure 2.7 wherein it may be seen that $V_1$, $V_2$, \ldots, $V_n$ are all bounded (as they are the outputs of opamps) amounting to $E$, as given in (2.39), having a defined lower bound.
The last term in (2.39) is usually neglected for high values of the open-loop gain of the opamp used to realize the neurons. The first term on the right hand side of (2.39) is quadratic which tries to minimize the number of colors. The second term has got a negative sign. Therefore, the energy function $E$ will be minimized if second term is maximized. This happens when the voltages corresponding to connected nodes in a graph are far away from each other. The first two terms on the right hand side are balancing each other to color a graph properly. The third term also contributes to lowering of number of different colors by eliminating all those local minima in the energy function for which node voltages are negative. The proposed network was tested for various random graphs using PSPICE simulations as well as breadboard implementation. The results of the tests are given in Table 2.1 from which it can be seen that the proposed network gives a solution to all the problems tested and in all the cases the solution is very near to the chromatic number of the graph. The performance of the proposed network was also compared with the NOSYNN-based voltage-mode graph colouring network proposed earlier [137]. Table 2.2 presents this performance comparison. It is evident that improvements have been achieved both in the best and the average solutions for most examples. The network was further tested on three standard benchmark problems for graph coloring [184, 35]. Simulation runs of the proposed network for the benchmarking problems are presented in Table 2.3 from where it can be seen that the best solution in each case is very near to the chromatic number of the corresponding graph.

It may be mentioned that monolithic integration of the graph colouring circuit of Figure 2.10 requires the fabrication of a large number of resistors which tends to consume a lot of chip area. A better alternative from the viewpoint of chip area conservation would be the use of transconductance elements, having voltage inputs and a current output, in the feedback paths. Conveying of the synaptic signals as currents reduces the overall number of resistances drastically and makes the circuit favorable from the viewpoint of actual VLSI
2.3 Chosen Problems: Description & Applications

Table 2.3: PSPICE simulation results for the proposed network applied to graph coloring benchmark problems

<table>
<thead>
<tr>
<th>S. No.</th>
<th>Benchmark Problem</th>
<th>Description (Nodes, Edges, Chromatic Number)</th>
<th>PSPICE Simulation Results</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>No. of colors</td>
<td>Frequency of occurrence</td>
</tr>
<tr>
<td>1</td>
<td>myciel3.col</td>
<td>11, 20, 4</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>myciel4.col</td>
<td>23, 71, 5</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>queen5_5.col</td>
<td>25, 320, 5</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>7</td>
</tr>
</tbody>
</table>

implementation. Such an implementation is referred to as a mixed-mode realization and is discussed in Appendix A.

The graph colouring and ranking networks presented in [137, 81], as well as the improved graph colouring networks presented in this section, demonstrate the validity of the NOSYNN architecture for application in real-world problems. Therefore, the NOSYNN architecture was selected as the starting point for obtaining neural circuits for other problems of interest like solving linear equations, linear programming and quadratic programming.

2.3 Chosen Problems: Description & Applications

This section deals with a description of the mathematical problems to which the various proposed implementations of the NOSYNN are applied. These include the solution of a system of linear equations, and the minimization of linear and quadratic objective functions subject to linear constraints, referred to as Linear and Quadratic Programming Problems respectively.
2.3 Chosen Problems: Description & Applications

2.3.1 System of Simultaneous Linear Equations

The solution of a system of linear equations, such as (2.46), has been a primary goal for computation since the time of the abacus. The earliest known references to word problems requiring the solution of a system of linear equations appear in ancient Babylonian texts dating back to *circa* 300 BC [62]. At that time, the Salamis Tablet, which was a ‘counting board’ and a rudimentary form of abacus, was probably used to solve the linear equations. Later, around 200 BC Chinese mathematicians put forward the method of ‘calculation by square tables’ for solving a system of simultaneous linear equations. This method appeared in Chapter Eight of the Chinese mathematical text *Jiu Zhang Suanshu* or The Nine Chapters on the Mathematical Art [182]. Although the nomenclature is not so illuminating, the method was essentially what we now know as Gaussian Elimination. After that there was little development in methods to solve linear equations for almost two millennia. In 1750, Gabriel Cramer proposed the Cramer’s rule for solving a set of $n$ simultaneous linear equations in $n$ variables. In 1810, Carl Friedrich Gauss devised a notation for symmetric elimination that was adopted in the 19th century by professional hand computers to solve the normal equations of least-squares problems [90]. Since then, other methods to solve a system of simultaneous linear equations that have evolved include Gauss-Jordan elimination, LU decomposition, Cholesky decomposition, Levinson recursion, etc. Another significant contribution from the viewpoint of solving large systems of linear equations on modern day computers came in 1966 when James Wilkinson proposed the iterative refinement method [131].

Babbage’s Analytical Engine (1836) was the first attempt at automating the equation solving process. It was followed by the Atanasoff-Berry Computer (1941) which marked the transition from mechanical to an electronic computing architecture [62, 170]. However, consuming almost a minute for each addition/multiplication, the Atanasoff-Berry Computer was certainly slow. The era of fast equation solving computers was signaled by FPS-164/MAX (1984).
It was followed by the ClearSpeed CSX600 (2005) which was capable of working at 25 Gflops/s consuming 10 Watts [62]. The ClearSpeed CSX600 (2005) capable of solution times of the order of microseconds is among the fastest available linear equation solvers. However, it needs to run a host operating system and is prone to “OS jitter” at high operating speeds. These issues together with the fact that its power consumption is around 10 Watts make the CSX600 unsuitable for real-time and/or portable applications. Among the current state of the art solutions, the recently developed concurrent multiprocessor based architectures for solving linear equations; only systolic/wavefront arrays [96] and the Block Data Parallel Architecture (BDPA) are suited for solving computationally intensive problems [173]. However, even with the extensive simulation tools developed for the BDPA, there is still a fundamental need to show the prospective power of such architectures. Such arrangements may not be suitable for real-time and/or portable applications where a dedicated, compact, low-power solution is desirable. This has led to research efforts being directed towards the development of specialized hardware for the solution of linear equations.

Another approach to solving linear equations was to use specialized machines. One such mechanical linear equation solver was constructed in 1936 at the Massachusetts Institute of Technology [172]. The late 1940’s witnessed the advent of electronic methods for solving linear equations [160]. Several more analog circuits were reported in the next decade [1, 115, 122]. In 1988, an analog resistive network was proposed for solving linear equations [77]. Later, neural networks promising massively parallel processing and fast convergence were applied to solve linear equations [80, 25, 161, 31, 32, 163, 185, 179, 83]. More recently, Field Programmable Gate Arrays (FPGAs) with their inherent capability to be used as multi-million-gate system-on-chip have been employed to parallelize the task of solving large systems of equations [165].

A general system of $m$ linear equations with $n$ unknowns can be written
2.3 Chosen Problems: Description & Applications

as

\[\begin{align*}
a_{11}x_1 + a_{1n}x_n + \ldots + a_{1n}x_n &= b_1 \\
a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n &= b_2 \\
    \vdots \\
a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n &= b_m
\end{align*}\]

where \(x_1, x_2, \ldots, x_n\) are the unknowns, \(a_{11}, a_{12}, \ldots, a_{mn}\) are the coefficients of the system of linear equations, and \(b_1, b_2, \ldots, b_m\) are the constant terms.

Often the coefficients and unknowns are real or complex numbers, but polynomials and elements of an abstract algebraic structure are also possible. In general, for the set of equations (2.46) to possess a unique solution, the linear equations must be linearly independent and \(m = n\).

Solving a system of simultaneous linear equations is one of the most fundamental problems in algebra. In the context of modern day systems, solving linear equations is an integral part of many scientific and engineering problems viz. curve fitting, electrical circuit analysis, multiple correlation as well as real time applications like real-time speech coding, image processing, stochastic modeling, and computer-aided realistic three-dimensional image synthesis [62, 94, 173]. Present day systems rely on software algorithms to arrive at the solution for a given set of linear equations. However computer algorithms, by virtue of their sequential nature, tend to have a solution time which may not be practical for real-time systems. Further, running of an algorithm may also require an operating system which may not be feasible on a portable system where a dedicated, compact, low-power solution is desirable. This has led to research efforts being directed towards the development of specialized hardware for the solution of linear equations. This thesis is an attempt to develop voltage-mode, mixed-mode and current-mode circuits for solving linear equations. Further, it has been shown that minor modifications in the proposed neural networks for solving linear equations enable them to solve two important constrained optimization problems viz. linear and quadratic programming [94].
2.3 Chosen Problems: Description & Applications

2.3.2 Linear Programming Problem

Mathematical programming, in general, is concerned with the determination of a minimum or a maximum of a function of several variables, which are required to satisfy a number of constraints. Such solutions are sought in diverse fields including engineering, operations research, management science, computer science, numerical analysis, and economics [94, 84].

A general mathematical programming problem can be stated as [84]:

\[
\text{Minimize } f(x)
\]

subject to

\[
g_i(x) \geq 0 \quad (i = 1, 2, \ldots, m) \tag{2.47}
g_j(x) = 0 \quad (j = 1, 2, \ldots, p) \tag{2.48}
\]

\[
x \in S \tag{2.49}
\]

where \(x = (x_1, x_2, \ldots, x_n)^T\) is the vector of unknown decision variables, and \(f, g_i (i= 1, 2, \ldots, m), h_j (j= 1, 2, \ldots, p)\) are the real-valued functions of the \(n\) real variables \(x_1, x_2, \ldots, x_n\).

In this formulation, the function \(f\) is called the objective function, and inequalities (2.47), equations (2.48) and the set restrictions (2.49) are referred to as the constraints. It may be mentioned that although the mathematical programming problem (MPP) has been stated as a minimization problem in the description above, the same may readily be converted into a maximization problem without any loss of generality, by using the identity

\[
\max f(x) = -\min [-f(x)] \tag{2.50}
\]

As a special case, if all the functions appearing in the MPP are linear in the decision variables \(x\), the problem is referred to as a linear programming problem (LPP). Such LPPs have been investigated extensively over the past decades, in view of their fundamental roles arising in a wide variety of engineering
and scientific applications, such as, pattern recognition [9], signal processing [33], human movement analysis [79], robotic control [186], and data regression [187]. Other real life applications include portfolio optimization [183], crew scheduling [20], manufacturing and transportation [129], telecommunications [28], and the TSP [30].

2.3.3 Quadratic Programming Problem

A quadratic programming problem (QPP) is a special case of the general MPP discussed in the previous section for which objective function $f$ is a second-order function of the decision variables while the constraints remain linear [84]. Moreover, optimization problems with nonlinear objective functions are usually approximated by a second-order system and then solved by standard quadratic programming techniques. Quadratic programming problems arise naturally in a variety of applications, such as structural analysis [12], optimal control [15], plastic analysis [112], antenna array pattern synthesis [128], geometric optimization [144], propulsion physics [21], multi-commodity networks [41], etc.

2.4 Overview of relevant literature

Since the re-emergence of NNs as a viable alternative for the solution of combinatorial optimization problems, several neural networks have been proposed to solve systems of linear equations, LPP and QPP. Jang, Lee and Shin proposed a neural network for matrix inversion which can be applied, with some modifications, to solve simultaneous linear equations [80]. Finding the inverse of a matrix was expressed as an optimization problem and the dissipative dynamics approach of Hopfield architecture was employed to design the electronic hardware. For an $n \times n$ matrix, this method uses $n$ essentially similar networks, each network optimizing an energy function. Chakraborty, Mehrotra, Mohan and Ranka improved upon the network of [80] and employed only a single energy function having a global minimum at a solution, if the chosen
system of linear equations is solvable [25]. However, with a convergence time to the order of ten seconds obtained during the course of simulations, their network was certainly slow.

Around the same time, Wang proposed an electronic realization of a recurrent neural network for solving linear equations [161]. To solve an $n$-variable system of equations, the network employs three operational amplifiers, one capacitor and $(n + 5)$ resistances to emulate a single neuron and the time to arrive at the solution is of the order of hundreds of milliseconds [161]. Cichocki & Unbehauen presented a neural circuit for solving linear equations which was able to provide a significantly improved solution time of the order of microseconds but at the cost of increased hardware complexity [31]. Each neuron in [31] comprises of three weighted summers and an inverting integrator. Wang and Li employed a linear activation function in their neural network for solving linear equations [163]. The electronic realization of the network presented in [163] is very similar to the one proposed in [161]. Zhang, Ganis and Sereno employed Anti-Hebbian synapses for the solution of linear equations [185]. Xia, Wang & Hung came up with a linear equation solver [179], which is essentially a generalized neural network based implementation of Censor and Elfving’s method for linear inequalities [24]. They used an approach similar to [161], utilizing weighted adders and integrators to realize the neurons. More recently, Jiang proposed a recurrent neural network for the on-line solution of linear equations with time varying variables [83].

LPP has also received considerable research attention from the neural networks community. The first solution of the linear programming problem was proposed by Tank and Hopfield wherein they used the continuous-time Hopfield network [155]. From the computational aspect, the operation of Hopfield network for an optimization problem, like the LPP, manages a dynamic system characterized by an energy function, which is the combination of the objective function and the constraints of the original problem [168]. However, the Hopfield network when applied to solve LPP fails to satisfy the Kuhn-Tucker optimality conditions for a minimizer. Over the years, the penalty function
approach has become a popular technique for solving optimization problems. Kennedy & Chua proposed an improved version of Tank & Hopfield’s network for LPP in which an inexact penalty function was considered [87]. The requirement of setting a large number of parameters was a major drawback of Kennedy & Chua’s LPP network [33]. Also, only approximate solutions were obtained as for true minimizations, the penalty parameter was required to be infinitely high which was impossible realistically. Rodriguez-Vazquez et al. used a different penalty method to transform the given LPP into an unconstrained optimization problem [140]. For solving a LPP in \( n \) variables with \( m \) constraints, their network employed \( n \) integrators, \( m \) summers, a constraint block comprising of \( m \) comparators and \((m + 1)\) AND gates. Although Rodriguez-Vazquez et al. later pointed out that their network had no equilibrium point in the classical sense [139], investigations by Lan et al. proved that the network can indeed converge to an optimal solution of the given problem from any arbitrary initial condition [97]. Maa & Shanblatt employed a two-phase neural network architecture for solving LPPs [110] but the network was certainly slow being able to provide solutions in times of the order of seconds. Chong et al. analyzed a class of neural network models for the solution of LPPs by dynamic gradient approaches based on exact non-differentiable penalty functions [29]. They also developed an analytical tool aimed at helping the system converge to a solution within a finite time. In the sample LPP solved in [29], the time taken by the network to arrive at the solution was around 600 milli-seconds. In an approach different from the penalty function methods, Zhu, Zhang and Constantinides proposed a Lagrange method for solving LPPs through Hopfield networks by employing two distinct types of neurons viz. the variable neurons and the Lagrangian neurons [189]. Instead of following a direct descent approach of the penalty function, the network searched for a first-order necessary condition of optimality in the state space. Xia and Wang used bounded variables to construct a new neural network approach to solve LPP with no penalty parameters. They suggested that the equilibrium point is the same as the exact solution when the primal and dual
problems are solved simultaneously [178]. However, only a block arrangement 
was provided in [178], and no actual implementation was suggested. More 
recently, Malek & Yari proposed two new methods for solving the LPP and pre-
sented optimal solutions with efficient convergence within a finite time [113]. 
Lastly, Ghasabi-Oskoei, Malek and Ahmadi have presented a recurrent neural 
network model for solving LPP based on a dynamical system using arbitrary 
initial conditions. The method does not require analog multipliers thereby 
reducing the system complexity [57]. However, in this case too, only a block 
arrangement is presented without details of the actual hardware realization. 
Furthermore, the solution time for the network is in seconds thereby making 
it unsuitable for real-time applications.

Various methods to solve QPP by employing neural network approaches 
are available in the technical literature. Kennedy & Chua extended the Tank 
and Hopfield network by developing a neural network for solving nonlinear 
programming problems, by satisfaction of the Karush–Kuhn–Tucker optimal-
ity conditions [87]. However, the need to set a penalty parameter means 
that the network can generate approximate solutions only and implementation 
problems arise when the penalty parameter is large. Each variable amplifier 
comprises of 2 opamps, 2 resistors and 1 capacitor whereas for satisfying each 
constraint, the constraint amplifier employs 3 opamps, 2 resistors and 1 diode 
[87]. Wang proposed a recurrent neural network for solving QPPs with equal-
ity constraints. The network is asymptotically stable and is able to generate 
optimal solutions to quadratic programs with equality constraints. An opamp 
based circuit realization of the network is also presented which requires \((n+m)\) 
neurons for solving a QPP in \(n\) variables with \(m\) constraints. Each neuron is 
made up of a summer, an integrator, and an inverter consuming 3 opamps, 
1 capacitor and \((n+5)\) resistors [162]. Wang’s network is not suitable for 
real time applications as it takes around 50 ms to arrive at the solution [162]. 
A rigorous analysis of the prominent neural networks for QPP, available till 
that time (1992), is presented in [110]. Forti & Tesi presented new conditions 
capable of ensuring existence, uniqueness, and global asymptotic stability of
the equilibrium point for Kennedy and Chua’s network [51]. Wu et al. proposed two neural network models for solving LPP and QPP, the convergence of which was not dependent on the network parameters [176]. Around the same time, Xia also put forward a neural network capable of solving both LPP and QPP in which no parameter tuning was necessary. Moreover, the actual hardware implementation was somewhat simplified, as compared to its contemporaries, because of the fact that no analog multipliers were required for the variables [177]. To solve a QPP in \( n \) variables with \( m \) constraints, Xia’s network consisted of \((2m^2 + 4mn)\) amplifiers, \((2m^2 + 4mn + 3m + 3)\) summers, \((n + m)\) integrators, and \( n \) limiters. Tao, Cao and Sun further simplified the network of Xia [177], and reduced the system complexity [156]. More recently, Liu and Wang presented a one layer feedback neural network with a discontinuous hard-limiting activation function for solving QPP in which the number of neurons is the same as the number of decision variables [104]. Each neuron in [104] is composed of 2 adders, \((3n + 1)\) resistors, 1 limiter and 1 integrator. Although significant reduction in circuit complexity is achieved, the time that the circuit takes to arrive at the correct solution is of the order of \( \text{seconds} \) thereby making the circuit unsuitable for applications requiring fast solution times. A comprehensive bibliography of the technical literature related to QPP can be found in [60].

2.5 Summary

A brief overview of the background information which is pertinent to the work contained in the thesis is presented. The popular Hopfield network is first explained and issues in convergence and other limitations associated with the network are highlighted. The NOSYNN, which was proposed as an alternative to the Hopfield network, is then explained and an application of the NOSYNN to graph coloring problems is also presented. Thereafter, the chosen problems for which the proposed hardware implementations of the NOSYNN have been tested, \( \text{viz.} \) the solution of a system of simultaneous linear equations and linear
2.5 Summary

& quadratic programming, are discussed. A survey of existing neural networks for the solution of the chosen problems is then presented.

From this point onwards, this thesis presents the proposed neural architectures for the solution of the chosen problems starting with exploring the possibility of solving linear equations using the Hopfield network.