Tunability of Auto Resonance Network

V M Aparanji[1]✉ (https://orcid.org/0000-0001-8030-6779), Uday V Wali[2] (https://orcid.org/0000-0003-1920-9990), R Aparna[3] (https://orcid.org/0000-0001-7330-867X)

[1] Dept. of E&CE, Siddaganga Inst. of Technology, Tumakuru, Karnataka, India
[2] Dept. of E&CE, KLE Dr MSS CET, Belagavi, Karnataka, India
[3] Dept. of ISE, Siddaganga Inst. of Technology, Tumakuru, Karnataka, India
{vma1508, udaywali, raparna27}@gmail.com

Abstract. We present a new type of Artificial Neural Network (ANN) called Auto-Resonance Network (ARN) derived from synergistic control of biological joints. The network can be tuned to any real valued input without any degradation of learning rate. Neuronal density of the network is low and grows at a linear or low order polynomial rate with input classification. Input coverage of the neuron can be tuned dynamically to match properties of input data. ARN can be used as a part of hierarchical structures to support deep learning applications.

Keywords: Artificial Neural Network, Auto Resonance Network, Self Organizing Maps, Adaptive Resonance Theory.

1 Introduction

Classical neural networks suffer from size and temporal superposition, generally called stability-plasticity dilemma in Artificial Neural Network (ANN) literature[1], [2], [3]. Neuroscience studies describe these effects as the binding problem and superposition catastrophe [4]. Some researchers take this a step further and state that the electrical oscillations in the biological sensory systems trigger the cells in a sequence, effectively serializing certain recognition activity in time, adding a new Degree of Freedom (DoF) to the biological recognition engine [5]. As the neural cells in a network do not increase over time, the size of the initial neural network has to be large enough to accommodate the likely size of knowledge that will be acquired over time. Each learning experience has to bind to a subset of the existing neural infrastructure. However, as the knowledge base increases, newer knowledge has to superimpose on existing infrastructure, possibly fragmenting the existing subsets. Old knowledge is replaced or distorted by new knowledge, effectively destabilizing the established subsets. This may not always have a damaging effect but does distort/refine old subsets.

Kohonen networks called the Self Organizing Maps (SOM) start with pre defined set of nodes initialized to random weights [6]. As the input is applied, some of the input nodes produce maximum output and one among them will be chosen as winner.

The key to Kohonen's networks is that the neighbors of the winner node adjust their weights towards that of the winner. Over period of time, repetition of this process creates a neighborhood of nodes that recognize similar inputs. Each of such neighborhoods represents one class of input. As the neighborhood is not constrained to be convex, it should be possible to support nonlinear classification. If the number of classes of inputs exceeds a certain number in relation to the total number of nodes in the network, the neighborhoods have to split and merge to accommodate new classes. Therefore, SOMs are subject to superposition catastrophe. Notice that they do not suffer from the binding problem as data classes are associated with sets of nodes.

On the other hand, neural networks based on Adaptive Resonance Theory (ART) [1] start with very few or no cells at all and add cells to the network when an input set cannot be recognized (classified). In these networks, a cell generates winning output when input matches its stored value. Other cells will have lesser output. Input range covered by a cell and the number of cells for a given range of values are both dynamically optimizable. Therefore, ART networks are minimal in terms of number of nodes required to build the network. Extensions of ART networks using real inputs and outputs, called ART2 scale and translate the input before storing in short term memory[1], [7]. However, the complexity of ART2 networks alters the simplicity and elegance of ART1 to the extent that these networks lose their relation with biological equivalents: A biological neuron would be simple, repetitive and connected.

In the following sections, a new type of neural network called Auto Resonance Network (ARN) is described. ARN can classify real valued multi-dimensional input and have an adjustable acceptance threshold (ρ) for each node in the network. These networks can be used as generic data classifiers by adding node labeling method or a neuronal layerand find applications in various areas of artificial intelligence.

2 Auto Resonance Network (ARN)

Auto Resonance Network will have a single layer of several neuronal nodes, each containing a pattern to match a particular input set. Each node has a specific stored pattern that is different from other nodes. At the basic level, behavior of the network would be as follows: Input will be applied as a set, one set at a time. Output layer would consist of a single layer of nodes all of which are connected to all inputs. Each node is tuned to recognize a specific pattern of input vector. Internal memory of the node may have an exact or approximate or transformed version of input it matches. When a new input is applied (i) one of the nodes is at resonance or (ii) some nodes are near resonance or (iii) none of the nodes are in resonance. In the first case it is the winner. In second case, the node with the highest output is selected as winner if the output is above a selection threshold. In all other cases, there is no recognition. When there is no recognition, a new node may be created such that it is tuned to match the input. Success of this network depends on finding a suitable function that offers good tunability and variable cover.

When a new node is appended to ARN, it is pre-tuned to resonate with current input. We will illustrate the concept with a single input network. Resonance of a node can be described using a simple equation

$$y_p = x(1 - x) \tag{1}$$

where, x is The input represented by a real number assumed to be normalized to a range of $\{0 \ldots 1\}$. Equation (1) will yield a maximum value of $1/4$ when the input $x = 1/2$, i.e., the node will resonate if the input is $1/2$. Therefore, we can use

$$y = 4(1 - x)x \tag{2}$$

to normalize the result to 1. In order to set the resonance at any value of $x_r \in \{0 \ldots 1\}$, we can scale the input such that

$$w x_r = 1/2 \quad or w = 1/(2x_r) \tag{3}$$

and calculate the output of the node as

$$y = 4 * (1 - wx)wx \tag{4}$$

The *resonant weight* w is computed when the node is inserted in the network and remains largely unaltered as a memory impression of the input x_r present at the time of creating the node.

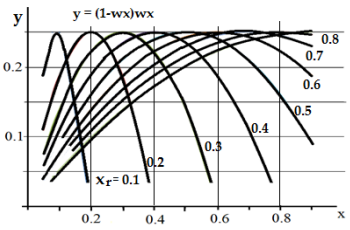


Fig.1. Resonance curves for various x_r

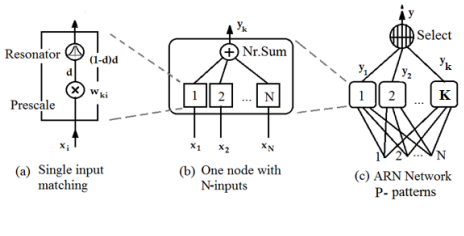


Fig. 2. Auto Resonance Network structure

The chart in Fig. 1 shows the behavior of nodes tuned at various points of resonance identified by x_r. Overall structure of ARN is shown in Fig. 2. In Fig. 2(a), scaling of single input by the resonant weight and computation of resonator output is shown. This is a basic module of ARN. A node with N input nodes is shown in Fig. 2(b). The input to a node consists of a vector

$$X = \{x_1, x_2, \ldots x_N\}, \quad x_i \in \{0 \ldots 1\}, i = 1 \ldots N \tag{5}$$

For each of the inputs, output of each resonator module given by (2) are summed and normalized as at the output of the node. A layer of ARN will have several such nodes as shown in Fig. 2(c). Assuming that there are K nodes, each one is tuned to a different input vector $X_k|_{t=t_k}$ where t_k is the time at which k-th node was created. We can extend (2) to describe output of a k-th node as

$$y_k = \frac{4}{N}\sum_i (1 - w_{ki}x_i)(w_{ki}x_i), \quad i = 1 \ldots N, k = 1 \ldots K \tag{6}$$

where, w_{ki} is the scaling factor for i-th input of k-th node. The resonant weight w_{ki} represents the in-the-node impression of input x_{ki} stored as memory in the node. Note that x_i is the i-th element of the input vector while x_{ki} represents the resonating input x_i for k-th node. This k-th node will produce maximum value of 1 when $x_i = \frac{1}{2w_{ki}}$, $i = 1 \ldots N$. In Fig. 2, the resonator function is indicated as $d(1 - d)$ where $d = wx$.

To summarize, each resonator corresponds to one input of one node. Each node has N inputs and same number of resonators. The output of a node is maximal when all

the resonators produce maximum output. Therefore, the resonant weight of a node is expressed as $W = 1/2X_r$. The resonant weights are calculated only once, when the node is added to the network. There will be only one node that maximally matches exact combination of inputs. For these reasons we call this network as Auto Resonance Network (ARN).

2.1 Envelop Functions

It can be seen from Fig. 1 that output of a resonating node decreases as a continuous function of input on either side of input values. If the output of a node is above a threshold value the node is a winner. By reducing or increasing the threshold the range of input values to which a node resonates can be adjusted. For example, if the threshold is reduced, the range increases and vice versa. Set of all inputs when the output of a node is above a threshold is called coverage of the node.

Coverage of k-th node can be expressed as

$$C_k = \{X \mid (y_k > \rho) \text{and} (y_k > y_i, \forall\, i \neq k)\} \quad (7)$$

where ρ is the threshold value.

We may further note from Fig. 1 that coverage of area for each node is not same for a given threshold. For example the peak for $X_r = 0.1$ is significantly sharper than the one at $X_r = 0.2$. We could set separate thresholds to individual nodes such that all nodes have similar coverage. However, a better way to correct this situation is to use a non-linear scaling of input. We call these functions as envelop functions.

Envelop functions can provide several advantages. For example, they can transform unbound input $x \in \mathbb{R}$ into bound region like $\{0 \ldots 1\}$. If the envelop function modifies the input $x_s = g(x)$ then, the resonant weights also should be scaled with identical function.

$$w_{ki} = \frac{1}{2\,g(x_{ki})} \quad (8)$$

$$y_k = \frac{4}{N} \sum_i w_{ki}\, g(x_i)\,(1 - w_{ki}\, g(x_i)) \quad (9)$$

Envelop functions stretch or compress a specific part of the input range in order to exemplify an area of interest. Effect of some of these functions is shown in Fig.3.

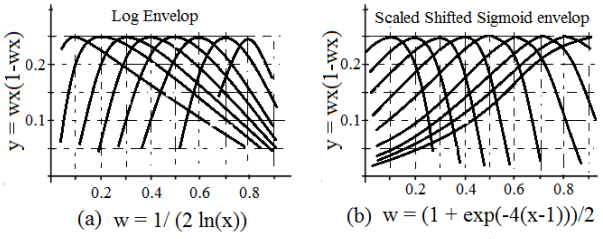


Fig.3. Effect of envelop functions on classification.

A simple scaled log function is shown in Fig 3(a) uses:

$$g(x) = \sigma \ln(x) \quad (10)$$

where σ is scaling factor. By adjusting σ we can effectively compensate for the non-linear coverage of (4). Fig. 3(a) shows the effect of (10) on coverage with $\sigma = 1$. A

modified sigmoid function shown in Fig 3(b) exhibits a controllable linear coverage. It uses an envelope function

$$g(x) = \frac{1}{(1+e^{-\sigma(x-1)})} \tag{11}$$

A value of $\sigma = 4$ is used for illustration. It is clear that envelop functions can reduce the non-uniform coverage across the input range.

2.2 Extending the Input Range

Though accepting input in the limited range of {0... 1} need not be a limitation, it would be convenient if there are other functions that provide a larger input range yet maintain the resonance property. Interestingly, there are many other monotonic functions to implement such resonance and build an ARN. A generic approach would be to define an additive inverse of the function over a range and multiply the two to get a resonance function. One such simple function is the difference function given by $(M_{ki} - x_i)$ such that

$$y_k = 1 + \sum_i (M_{ki} - x_i)(x_i - M_{ki}) \tag{12}$$

where M is the memory copy of the tuned input.

Another good candidate is the Scaled and Shifted Sigmoid Function (3SF) given below:

$$y_s = \frac{1}{(1+e^{-\sigma(x-M)})} \tag{13}$$

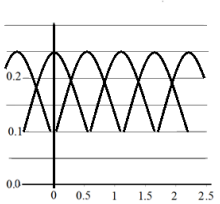
will map a real number in the range of $-\infty : \infty$ to a monotonically increasing value in the range 0:1. Interestingly, the function has a value of 0.5 at $x = M$. Therefore, we could replace $w_{ki}x_i$ in (4) with this function. It may be noted that $(1 - y_s)$ can be easily computed as

$$y_{s-} = \frac{1}{(1+e^{\sigma(x-M)})} \tag{14}$$

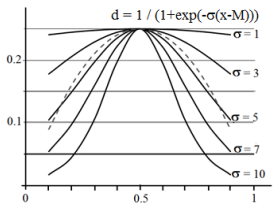
Therefore, for N = 1, we can rewrite (2) as

$$y = 4\left[\frac{1}{(1+e^{-\sigma(x-M)})}\right]\left[\frac{1}{(1+e^{\sigma(x-M)})}\right] \tag{15}$$

The sigmoid nature of this curve fits well with a physical neuron activation which shows saturation as the input increases, rather than growing monotonically. Equation (15) allows a node to be set to resonate at any $x_i \in \mathbb{R}$. Note that M can be used to select the point of resonance while σ can be used to control the tuning and hence coverage of the node as shown in Fig 4(a) and (b) respectively. Equation (15) provides a generalized function for implementing ARN nodes, albeit with increased complexity. The tradeoff between complexity versus flexibility can tilt towards (4) or (15) depending on end use.



(a) Shifted Sigmoid Resonators



(b) Tunability of sigmoid resonator

Fig. 4. Tunability of ARN using (15).

2.3 Tuning the nodes and coverage

Coverage of a node acts like noise margin by providing near maximal output when the input is close to the resonating value. Therefore, each node can recognize noisy input if it is within the coverage area. Coverage of the node can vary dynamically. Node may slowly shift to a different tuning point depending on the statistical properties of incoming data but stays close to the original tuned location.

Controlling the quality of resonance as shown in Fig. 4(b) can be used to sharpen the performance of ARN nodes. For example, if a node receives exactly the same input repeatedly, we can increase its sharpness by increasing σ. On the other hand, if the input value varies around the resonance value but within a threshold, we can reduce the value of σ to increase its coverage. A simple relation that can be used to tune the resonance is given by,

$$\sigma_{(n+1)} = \sigma_n + \eta\kappa f/(1 - \nu) \tag{16}$$

where, η is learning rate, κ is a proportionality constant related to f statistical frequency, number of times the node matched the input and ν is related to signal variance. Therefore, this equation provides basis for reinforcement learning on ARN nodes.

Stable nodes can undergo further tuning to increase or decrease the area covered by the nodes. This can be achieved by varying the selection threshold or the σ value associated with the node. This requires that the nodes compute statistical moments as they are accessed. For a node described by (15), and knowing that $y_{max} = 1$ and assuming $x_r = 0$, we can write the value of x for threshold of $y = \rho$ as,

$$\rho = \frac{4}{N}\left\{\frac{1}{(1+e^{-\sigma x})(1+e^{\sigma x})}\right\} \tag{17}$$

which gives an expression for coverage of an ARN node as a function of threshold and tuning factor. Eqn. (17) can be rewritten as

$$x_\rho = \frac{1}{\sigma}\cosh^{-1}\left(\frac{2}{N\rho} - 1\right) \tag{18}$$

which gives the coverage of an ARN node for various values of threshold and scale factor around the peak value.

2.4 Types of ARN Nodes

Typically, ARN nodes are created when an input does not produce resonance in existing nodes and the expected value of output is known. Such nodes have a well defined point of resonance, output mapping and adjustable coverage. We will call these as Type-1 nodes. Additional nodes can be created in absence of input by interpolating properties derived from Type-1 nodes. The output and associated data can be estimated as a perturbation of values of Type-1 nodes or interpolated using piecewise linear or other suitable approximation. We will call them as Type-2 nodes.

3 Results

ARN can perform real-world input classification. Figures5 and 6 show the Pattern classification using scaled shifted sigmoid envelop function for different thresholds. Each bubble indicates an input. Each color represents a node. If a node resonates on application of an input it is represented by the color of resonating node and shown at the location specified by input. If there is no resonating node, then the network is not resonating. In such a case a new node is created and appended to existing network. In other words, a new node is added when maximum output value of all nodes in the recognition layer is below the threshold. The node is assigned a color from a fixed list of colors. Therefore, color of the node depends on the order in which an input arrives during training and may vary from run to run. Group of inputs that drive the same node towards resonance (represented by same color) represent the coverage of the node. Increasing the threshold from 0.7 to 0.9 has increased the number of output nodes.

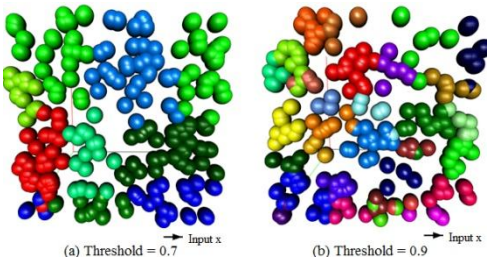


Fig.5.2D Pattern classification for different thresholds.

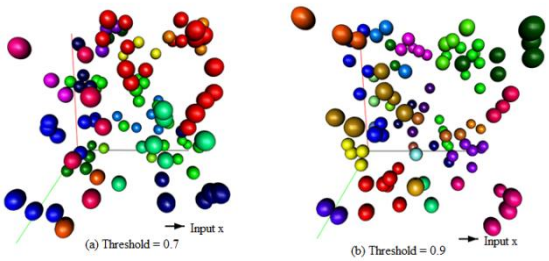


Fig. 6.3D Pattern classification for different thresholds

Deep learning methods for different applications are reported in [8]. It is possible to use several of the existing types of neural networks discussed in [9], [10] for path planning of robotic motion. A hierarchical network for path planning built using ARN has been reported in [11], [12].

4 Conclusions

The sparseness of ARN and the simplicity of resonance equations can make ARN suitable for implementations in embedded systems.An important advantage of ARN is ease of control over coverage and sparse node assignment. It is obvious that the order in which input is applied has a strong effect on how the ARN network gets created.

But nodes that are responsible for a specific output can be identified. The network can be refined by successively applying new data that covers the labeled data class to increase the accuracy of classification. Therefore, it is easy to see how the network has interpreted the data. A single layer ARN can classify convex data sets and therefore require labeling at higher levels of hierarchy in a typical deep learning neural network. ARN can be used in lower levels of such structures to provide data classification. Efforts are on the way to use this structure in various areas of current research.

References

1. Stephen Grossberg (1987) Competative Learning – From Interactive to Action to Adaptive Resonance, Cognitive Science 11, pp 23-63
2. C. von der Malsburg (1987) Synaptic Plasticity As Basis Of Brain Organization, The Neural and Molecular Bases of Learning, John Wiley & Sons Limited, pp 1-24
3. Christoph von der Malsburg (1999) The What and Why of Binding, The Modeler's Perspective, Open Archive, Elsevier, Vol. 254, Iss.1, pp. 95-104
4. Thomas Burwick (2006) Oscillatory Networks: Pattern Recognition without a Superposition Catastrophe, Neural Computation, Vol. 18, Issue. 2, pp. 356-380
5. Valerie Gray Hardcastle (1996) The Binding Problem and Neurobiological Oscillations, Chapter. 4, Towards a Science of Consiousness, First Tuscon Discussions and Debates, Ed. Stuart R. Hameroff, Alfred W. Kaszniak, Alwyn C. Scott, A Bradford book, The MIT Press, Cambridge, Massachusets
6. T Kohonen (1990) The Self-organizing Map, Invited paper, Proceedings of the IEEE, Vol. 78, No 9, pp 1464-1480
7. S. Hochreiter and J Schmidhuber(1997) The Long Short-Term Memory, Neural Computation, Vol 9, No. 8, pp. 1735-1780
8. He, H., McGinnity, T.M., Coleman, S., Gardiner B.(2014) Linguistic Decision Making for Robot Route Learning, IEEE Trans. Neural Networks And Learning Systems, Vol. 25, No. 1:203-215
9. Aparanji, V.M., Wali, U.V., Aparna, R.(2016) A Novel Neural Network Structure for Motion Control in Joints, ICEECCOT Mysore, 227-232, also available from IEEE Xplore digital library http://ieeexplore.ieee.org/document/7955220/
10. Aparanji, V.M., Wali, U.V., Aparna, R. (2017) Robotic Motion Control using Machine Learning Techniques, 6th IEEE International Conference on Communication and Signal Processing, Melmaravattur, (ICCSP 2017) 1241-1245, also available from IEEE Xplore digital library
11. Aparanji, V.M., Wali, U.V., Aparna, R. (2017) Automated Path Search and Optimization of Robotic Motion Using Hybrid ART-SOM Neural Networks, International Conference on Recent Advancement in Computer and Communication, Bhopal, (ICRAC-2017), Springer LNNS, 415-423, Web page Available at https://doi.org/10.1007/978-981-10-8198-9_43
12. Aparanji, V.M., Wali, U.V., Aparna, R. (2018) Robotic Motion Control using Machine Learning Techniques, International Conference on Conference on Cognitive Computing & Information Processing, Bangalore, (CCIP 2017) Springer CCIS801, 386–394, Web page Available at doi.org/10.1007/978-981-10-9059-2_34, Springer CCIS, /doi.org/10.1007/978-981-10-9059-2_34, pp., 2018