

University of Dayton eCommons

Electrical and Computer Engineering Faculty
Publications

Department of Electrical and Computer
Engineering

4-2015


Gaussian Weighted Neighborhood Connectivity of Nonlinear Line Attractor for Learning Complex Manifolds

Theus H. Aspiras
University of Dayton

Vijayan K. Asari
University of Dayton, vasari1@udayton.edu

Wesam Sakla
Air Force Research Laboratory

Follow this and additional works at: https://ecommons.udayton.edu/ece_fac_pub

 Part of the [OS and Networks Commons](#), [Systems and Communications Commons](#), and the [Theory and Algorithms Commons](#)

eCommons Citation

Aspiras, Theus H.; Asari, Vijayan K.; and Sakla, Wesam, "Gaussian Weighted Neighborhood Connectivity of Nonlinear Line Attractor for Learning Complex Manifolds" (2015). *Electrical and Computer Engineering Faculty Publications*. 386.
https://ecommons.udayton.edu/ece_fac_pub/386

This Conference Paper is brought to you for free and open access by the Department of Electrical and Computer Engineering at eCommons. It has been accepted for inclusion in Electrical and Computer Engineering Faculty Publications by an authorized administrator of eCommons. For more information, please contact frice1@udayton.edu, mschlangen1@udayton.edu.

Gaussian Weighted Neighborhood Connectivity of Nonlinear Line Attractor for Learning Complex Manifolds

Theus H. Aspiras^a, Vijayan K. Asari^a Wesam Sakla^b

^aUniversity of Dayton, 300 College Park, Dayton, USA;

^bAir Force Research Lab, Sensors Directorate, WPAFB, Fairborn, USA

ABSTRACT

The human brain has the capability to process high quantities of data quickly for detection and recognition tasks. These tasks are made simpler by the understanding of data, which intentionally removes redundancies found in higher dimensional data and maps the data onto a lower dimensional space. The brain then encodes manifolds created in these spaces, which reveal a specific state of the system. We propose to use a recurrent neural network, the nonlinear line attractor (NLA) network, for the encoding of these manifolds as specific states, which will draw untrained data towards one of the specific states that the NLA network has encoded. We propose a Gaussian-weighted modular architecture for reducing the computational complexity of the conventional NLA network. The proposed architecture uses a neighborhood approach for establishing the interconnectivity of neurons to obtain the manifolds. The modified NLA network has been implemented and tested on the Electro-Optic Synthetic Vehicle Model Database created by the Air Force Research Laboratory (AFRL), which contains a vast array of high resolution imagery with several different lighting conditions and camera views. It is observed that the NLA network has the capability for representing high dimensional data for the recognition of the objects of interest through its new learning strategy. A nonlinear dimensionality reduction scheme based on singular value decomposition has found to be very effective in providing a low dimensional representation of the dataset. Application of the reduced dimensional space on the modified NLA algorithm would provide fast and more accurate recognition performance for real time applications.

1. INTRODUCTION

The human brain has an incredible ability to process high volumes of data and provide outputs to interact with the body and its surroundings. Through several synaptic junctions, the neurons in the brain activate to discern different inputs that come in and activate other neurons to allow a response for other neurons. These neurons are interconnected in a large web of connections, which fire in different ways. To represent the activation and training of the brain, we use neural networks as the best model.

There several types of neural networks that are available. We have the feed-forward neural network,¹ which is able to propagate activations to different layers in the network, thus creating an output at the end. These types are suitable for different recognition tasks,² function approximation,³ and data compression.⁴ There are also recurrent neural networks, which propagates outputs to the same layer of the network. This uses a stability criteria to take inputs and converge towards a specific trained pattern. Recurrent neural networks can be used for modeling,⁵ optimization,⁶ and noise reduction.⁷

A recurrent neural network that was discovered was the Hopfield network,⁸ which converges towards a desired trained state due to the energy function to reduce the error. It has the ability to store several orthogonal patterns and recall these patterns, even with the presence of noise. This type is a point attractor network, which attracts an input towards a certain trained pattern. There are also line attractors and oscillatory attractors which can attract to certain types of patterns. The nonlinear line attractor (NLA) network, formed by Seow et al.⁹ is the type we will be focusing on, which has applications in skin color association, pattern association,¹⁰ and pose and expression invariant face recognition.¹¹

Further author information: (Send correspondence to V.K.A.)

T.H.A: E-mail: aspirast1@udayton.edu, Telephone: 1 937 229 1774

V.K.A: E-mail: vasari1@udayton.edu, Telephone: 1 937 229 4504

W.S.: E-mail: wesam.sakla@us.af.mil, Telephone: 1 937 528 8582

Most recurrent neural networks are totally interconnected, meaning that every node in the network is connected to each other. If we glean information about the structures in the brain, we can see that the neurons are not fully interconnected, but connected only to surrounding neurons. These types of structures introduce modularity to the network, thus different regions of the network must be able to interact with each other. Happel et al.¹² investigated modularity in neural networks and found that it increase the recognition in different tasks. Even to reduce the complexity of a model, we can use modularity to help recognition in certain regions to help in the recognition in the whole task.¹³

If we incorporate this type of structure with the NLA network, we should be able to reduce the training time for the network, since the amount of interconnections are less. We can also see an increase in the recognition rates and a decrease in convergence times due to the modularity reducing redundancies in computation.

The main contributions of this paper is:

- Gaussian weighting to the Nonlinear Line Attractor Network
- Reduction of computational complexity for the NLA

2. METHODOLOGY

Let the response x_i of the i^{th} neuron due to the excitations x_j from other neurons for the s^{th} pattern in a fully connected recurrent neural network with n neurons be expressed as:

$$x_i^s = \frac{1}{N} \sum_{j=1}^N \Lambda_i(x_j^s) \text{ for } 1 \leq i \leq N \quad (1)$$

where Λ is defined by a k^{th} order nonlinear line as:

$$\Lambda_i(x_j^s) = \sum_{m=0}^k w_{(m,ij)}^s (x_j^s)^m \text{ for } 1 \leq i, j \leq N \quad (2)$$

The m^{th} order term of the resultant memory w^s can be expressed as:

$$w_m^s = \begin{pmatrix} w_{(m,11)}^s & \cdots & w_{(m,1N)}^s \\ \vdots & \ddots & \vdots \\ w_{(m,N1)}^s & \cdots & w_{(m,NN)}^s \end{pmatrix} \text{ for } 0 \leq m \leq k \quad (3)$$

To calculate the weights, we can use the above equation. The least squares estimation approach is able to calculate the best fit line using the polynomial method. To minimize the least squares error in the weight matrix, we can formulate the following equation.

$$E_{ij}[w_{(0,ij)}, w_{(1,ij)}, \dots, w_{(k,ij)}] = \sum_{s=1}^P [x_s^i - \Lambda_i(x_j^s)]^2 \text{ for } 1 \leq i, j \leq N \quad (4)$$

To minimize the least squares error, we must find that the derivative of the error with respect to the weight should be zero, as shown in the following equation.

$$\frac{\delta E_{ij}}{\delta w_{(m,ij)}} = 0 \text{ for each } m = 0, 1, \dots, k \quad (5)$$

We can then find that the equation can be reduced to a set of equations based on the order of the polynomial, as shown in the following equation.

$$\begin{aligned}
& w_{(0,ij)} \sum_{s=1}^P (x_j^s)^0 + w_{(1,ij)} \sum_{s=1}^P (x_j^s)^1 + \dots \\
& \quad + w_{(k,ij)} \sum_{s=1}^P (x_j^s)^k = \sum_{s=1}^P y_i^s (x_j^s)^0 \\
& w_{(0,ij)} \sum_{s=1}^P (x_j^s)^1 + w_{(1,ij)} \sum_{s=1}^P (x_j^s)^2 + \dots \\
& \quad + w_{(k,ij)} \sum_{s=1}^P (x_j^s)^{k+1} = \sum_{s=1}^P y_i^s (x_j^s)^1 \\
& \quad \vdots \\
& w_{(0,ij)} \sum_{s=1}^P (x_j^s)^k + w_{(1,ij)} \sum_{s=1}^P (x_j^s)^{k+1} + \dots \\
& \quad + w_{(k,ij)} \sum_{s=1}^P (x_j^s)^{2k} = \sum_{s=1}^P y_i^s (x_j^s)^k
\end{aligned} \tag{6}$$

Given that there is a nonlinear line that models the relationship between inputs, there must be another modeling of the variances of the data. This is done by creating an activation function, as shown in equation 7.

$$\Phi \{ \Lambda [x_j(t)] \} = \begin{cases} x_i(t) & \text{if } \psi_{ij}^- \leq \{ \Lambda_i [x_j(t) - x_i(t)] \} \leq \psi_{ij}^+ \\ \Lambda_i [x_j(t)] & \text{otherwise} \end{cases} \tag{7}$$

where

$$\Lambda_i(x_j(t)) = \sum_{v=0}^k w_{(v,ij)} (x_j(t))^v \tag{8}$$

These threshold regions can be expressed as:

$$\psi_{ij}^- = \begin{cases} \psi_{1,ij}^- & \text{if } 0 \leq x_j < \frac{L}{\Omega} \\ \psi_{2,ij}^- & \text{if } \frac{L}{\Omega} \leq x_j < \frac{2L}{\Omega} \\ \vdots & \\ \psi_{\Omega,ij}^- & \text{if } (\Omega - 1) \frac{L}{\Omega} \leq x_j < L \end{cases} \tag{9}$$

$$\psi_{ij}^+ = \begin{cases} \psi_{1,ij}^+ & \text{if } 0 \leq x_j < \frac{L}{\Omega} \\ \psi_{2,ij}^+ & \text{if } \frac{L}{\Omega} \leq x_j < \frac{2L}{\Omega} \\ \vdots & \\ \psi_{\Omega,ij}^+ & \text{if } (\Omega - 1) \frac{L}{\Omega} \leq x_j < L \end{cases} \tag{10}$$

Figure 1 shows how the weights are interconnect through the inputs.

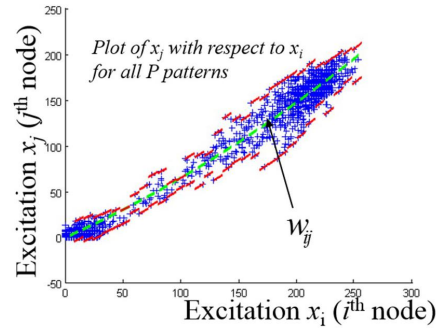


Figure 1. Interconnection of weights. The green line captures the nonlinear line modeling and the red lines capture the variances of the data

2.0.1 Nonlinear Dimensionality Reduction

To reduce the size and the computation time of the weight sets, we can reduce the dimensionality of the weights. Since the weights already have embedded the manifolds, we can use the weights in the reduction process. Given that there are r different line attractor networks, there will be y different outputs, as shown in the following equation.

$$\begin{aligned}
 Y_1 &= W_{1,k}X^k + W_{1,k-1}X^{k-1} + \dots + W_{1,0}X^0 \\
 Y_2 &= W_{2,k}X^k + W_{2,k-1}X^{k-1} + \dots + W_{2,0}X^0 \\
 &\vdots \\
 Y_r &= W_{r,k}X^k + W_{r,k-1}X^{k-1} + \dots + W_{r,0}X^0
 \end{aligned} \tag{11}$$

Each m^{th} term of the networks' memory is evaluated using singular value decomposition (SVD), as shown in the following equation.

$$w_m^d = U_m^d \Sigma_m^d (V_m^d)^T \text{ for } 0 \leq m \leq k \text{ and } 1 \leq d \leq r \tag{12}$$

The projection of the N -dimensional data to a z -dimensional subspace using a $z \times N$ sub-matrix obtained from the V -matrix of the SVD yields a z -dimensional output Y'_m where $z \ll N$.

2.0.2 Computational Strategy

We can reduce the amount of computations in the network so that training the network will take much less time. Given equation 6, we can compute certain portions of the data to reduce redundancies in the calculations. The following stages are given below.

Stage 1 would be the calculation of powers for the inputs. In equation 6, we see that every x_j^s has an order next to it, and this would be redundant for all different combinations of inputs and outputs and even inside the equations, since there are multiple terms with the same order, hence the same value.

Stage 2 would be the calculation of the weights, given the set of normal equations and using the values obtained from stage 1. This stage will take the most time due to the amount of data, since the weight matrix is $\#inputs \times \#inputs \times order$.

Stage 3 would be the calculation of the activation function, which is about the same as the calculation of stage 2. Since all of the orders are known, we can reduce the amount of computations needed for finding the order summation.

Stage 4 would be the calculation of the nonlinear dimensionality reduction. This step is dependent on the number of inputs and the order of the system.

| | | | | | | | | | | | | |
|---|---|----|----|-----|-----|-----|-----|-----|----|----|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 2 | 3 | 4 | 3 | 2 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 4 | 9 | 15 | 18 | 15 | 9 | 4 | 1 | 0 | 0 |
| 0 | 1 | 4 | 13 | 29 | 48 | 57 | 48 | 29 | 13 | 4 | 1 | 0 |
| 0 | 2 | 9 | 29 | 67 | 111 | 131 | 111 | 67 | 29 | 9 | 2 | 0 |
| 1 | 3 | 15 | 48 | 111 | 183 | 216 | 183 | 111 | 48 | 15 | 3 | 1 |
| 1 | 4 | 18 | 57 | 131 | 216 | 255 | 216 | 131 | 57 | 18 | 4 | 1 |
| 1 | 3 | 15 | 48 | 111 | 183 | 216 | 183 | 111 | 48 | 15 | 3 | 1 |
| 0 | 2 | 9 | 29 | 67 | 111 | 131 | 111 | 67 | 29 | 9 | 2 | 0 |
| 0 | 1 | 4 | 13 | 29 | 48 | 57 | 48 | 29 | 13 | 4 | 1 | 0 |
| 0 | 0 | 1 | 4 | 9 | 15 | 18 | 15 | 9 | 4 | 1 | 0 | 0 |
| 0 | 0 | 0 | 1 | 2 | 3 | 4 | 3 | 2 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |

Figure 2. An example of a 13 x 13 Gaussian kernel

2.1 Gaussian Nonlinear Line Attractor (GNLA) Network

The Gaussian Nonlinear Line Attractor Network is a modification to the original NLA network, which uses a neighborhood approach to improve the algorithm. Local information is more important than distant information, when looking at a biological perspective, so we can assume that using this architecture for the NLA algorithm would improve run times and classification ability.

When implementing a Gaussian neighborhood approach, we can change the coefficient in the front and add the distance equation. The equation can then be modified as:

$$x_i^s = \sum_{j=1}^N \alpha_{ij} \Lambda_i(x_j^s) \text{ for } 1 \leq i \leq N \quad (13)$$

where

$$\alpha_{ij} = \exp\left(-\left(\frac{(x_i - x_j)^2}{2\sigma_x^2} + \frac{(y_i - y_j)^2}{2\sigma_y^2}\right)\right) \quad (14)$$

Instead of using the Gaussian Function, we can use the Gaussian kernel, for example a 13x13 Gaussian kernel as shown in Figure 2:

Instead of implementing just the coefficients, we can use this kernel for each neuron, which will effectively reduce the computation time. We can then change the equation

$$x_i^s = \sum_{j=1}^N A_{ij} \Lambda_i(x_j^s) \text{ for } 1 \leq i \leq N \quad (15)$$

Where n is the size of the kernel and

$$A_{ij} = \exp\left(-\left(\frac{(x_i - x_j)^2}{2\sigma_x^2} + \frac{(y_i - y_j)^2}{2\sigma_y^2}\right)\right) \quad (16)$$

We can also effectively reduce the computation time of the kernel by not computing the portions that contain zeros, as shown in Figure 3. According to the Gaussian kernel above (which is a 13x13), we can reduce the computation time even further since there are zeros in the kernel (28% of the Gaussian kernel are zeros).

| | | | | | | | | | | | | |
|---|---|----|----|-----|-----|-----|-----|-----|----|----|---|---|
| | | | | | 1 | 1 | 1 | | | | | |
| | | | 1 | 2 | 3 | 4 | 3 | 2 | 1 | | | |
| | | 1 | 4 | 9 | 15 | 18 | 15 | 9 | 4 | 1 | | |
| | 1 | 4 | 13 | 29 | 48 | 57 | 48 | 29 | 13 | 4 | 1 | |
| | 2 | 9 | 29 | 67 | 111 | 131 | 111 | 67 | 29 | 9 | 2 | |
| 1 | 3 | 15 | 48 | 111 | 183 | 216 | 183 | 111 | 48 | 15 | 3 | 1 |
| 1 | 4 | 18 | 57 | 131 | 216 | 255 | 216 | 131 | 57 | 18 | 4 | 1 |
| 1 | 3 | 15 | 48 | 111 | 183 | 216 | 183 | 111 | 48 | 15 | 3 | 1 |
| | 2 | 9 | 29 | 67 | 111 | 131 | 111 | 67 | 29 | 9 | 2 | |
| | 1 | 4 | 13 | 29 | 48 | 57 | 48 | 29 | 13 | 4 | 1 | |
| | | 1 | 4 | 9 | 15 | 18 | 15 | 9 | 4 | 1 | | |
| | | | 1 | 2 | 3 | 4 | 3 | 2 | 1 | | | |
| | | | | | 1 | 1 | 1 | | | | | |

Figure 3. An example of a 13 x 13 Gaussian kernel with zeros removed

2.1.1 Nonlinear Dimensionality Reduction

When calculating the weight matrices for the Gaussian NLA network, none of the weights contain any information of the Gaussian distances. To incorporate the function inside the weights, we must embed the terms into the weights by multiplying the normalized mask into the weights. Given equation 16, we can embed the normalized coefficients to multiply the weights using the following equation.

$$a^s = \begin{pmatrix} a_{(11)}^s & \cdots & a_{(1N)}^s \\ \vdots & \ddots & \vdots \\ a_{(N1)}^s & \cdots & a_{(NN)}^s \end{pmatrix} \quad (17)$$

The resultant multiplication of the weight set is given by the equation below.

$$\bar{w}_m^s = \begin{pmatrix} \bar{w}_{(m,11)}^s & \cdots & \bar{w}_{(m,1N)}^s \\ \vdots & \ddots & \vdots \\ \bar{w}_{(m,N1)}^s & \cdots & \bar{w}_{(m,NN)}^s \end{pmatrix} \text{ for } 0 \leq m \leq k \quad (18)$$

This nonlinear dimensionality reduction technique using SVD only focuses on a specific order of the weight matrices. There is no interconnection between the orders, in which the algorithm contains a summation between the orders to results in the final outputs. We propose that instead of reducing this dimensionality by adding the orders of the matrix for the results, we can keep the orders separate concatenate all of the orders, which are multiplied by the input, to give a larger but more meaningful distinction between the orders. This will allow even better recognition results using fewer total weights, even though the resulting vector can be potentially larger.

2.1.2 Complexity

In Stage 1, we can find that it will be the same complexity as the previous algorithm since we are just computing the powers.

In Stage 2, the original complexity is $\#inputs \times \#inputs \times order^2$ due to the linear solve algorithm, but modified complexity is $\#inputs \times \#neighbors \times order^2$, where $\# neighbors$ is significantly smaller than the size of the network. For example, given that we have a network of 60x80, which is 4800, an order of 4 for the polynomial, and a kernel of size 13×13 , the computation time will be 3.52% the computation time of the original. If we reduce the kernel by taking out all zeros in the function, the computation will be 2.52% the computation time of the original and 71.6% the computation time of the original kernel.

Table 1. The training times of all the different stages of the algorithms. it is found that the kernel algorithm without using zeros provides faster training times than all of the other algorithms.

| Runtime | Original | Current | Kernel | Kernel (No zeros) |
|---------------|------------|-----------|-----------|-------------------|
| Stage 1 | N/A | 15.2 min | 15.2 min | 15.2 min |
| Stage 2 | 1280 min | 95.3 min | 3.35 min | 2.4 min |
| Stage 3 | 87.5 min | 87.5 min | 3.08 min | 2.21 min |
| Stage 4 | 10.8 min | 10.8 min | 10.8 min | 10.8 min |
| Total Runtime | 1378.3 min | 208.8 min | 32.43 min | 30.61 min |

In Stage 3, the complexity should be reduced just as stage 2. In Stage 4, the complexity should be the same as the previous algorithm.

Table 1 shows the different run times for each stage of the algorithm. Going from the original to the current architecture, we see a large reduction in stage 2, which contains several redundancies in the data. Then using the kernel functionality, the run time for stage 2 is reduced even further, due to the algorithm only calculating regions with the kernel. The run time is further reduced when calculating without the zeros in the kernel.

3. RESULTS

The dataset is a EO Synthetic Vehicle Database, which contains several different cars, various lighting conditions, and multiple viewing angles, as shown in Figure 4. This dataset has been created to test the algorithms ability to model an entire vehicle manifold across these different scenarios. The following testing scenario has been setup using the EO Synthetic Vehicle Database. For training and testing data, we have limited the dataset to one specific lighting scenario and 120 viewpoints of three different vehicles (Avalon, Camry, Civic). We also scale the imagery to a 60x80 image for faster processing times. For dimensionality reduction, we use singular value decomposition and then use euclidean distance of the transformed data in comparison to the trained data for recognition of specific vehicles.

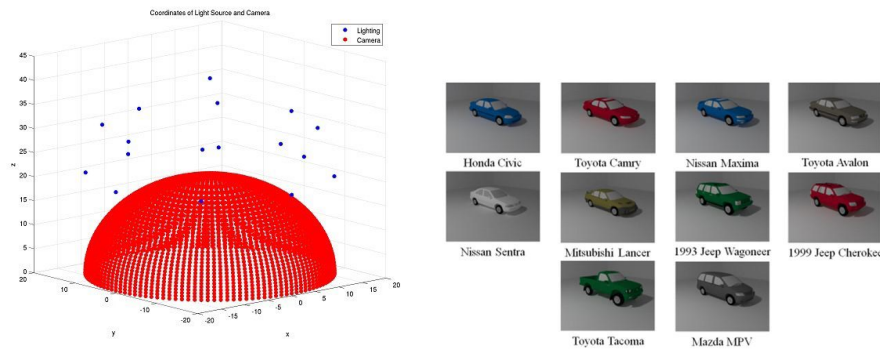


Figure 4. The EO Synthetic Vehicle Database. The top image contains the different camera views and lighting schemes and the bottom image contains the different vehicles in the database.

In Table 2, we can see that the Gaussian nonlinear line attractor has the best recognition results across all principal components. Even in small numbers of principal components, the GNLA network still is able to have the best recognition results for this database against both kernel principal component analysis (PCA) and the original NLA network. This is due to the modularity in the network. We can also see that the network has a run time much less than the original NLA architecture. By incorporating both the Gaussian weighting concept and the computational strategy, we are able to create a network that is better in most aspects.

4. CONCLUSION

It is observed that the Gaussian Nonlinear Line Attractor (GNLA) network has been drastically improved from the previous architecture. By introducing modularity to the network, we are able to reduce the computational run time, improve convergence characteristics, and increase the recognition rates. It has been seen in the results that

Table 2. The recognition results using the nonlinear line attractor network with the reduced dimensionality. It is found that the GNLA network produced the best results.

| # Prin. Comp. | Kernel PCA | NLA | GNLA |
|---------------|------------|-------|-------|
| 1 | 39.81 | 47.50 | 35.83 |
| 2 | 50.16 | 59.72 | 52.78 |
| 3 | 53.39 | 86.67 | 63.06 |
| 4 | 62.52 | 61.39 | 76.94 |
| 5 | 63.10 | 63.89 | 81.67 |
| 6 | 66.75 | 69.17 | 85.00 |
| 7 | 65.43 | 66.94 | 86.39 |
| 8 | 68.61 | 75.28 | 87.22 |
| 9 | 68.61 | 72.22 | 91.39 |
| 10 | 68.72 | 78.61 | 91.94 |
| All | 98.36 | 99.19 | 99.44 |

the GNLA architecture works well in the EO Synthetic Vehicle Database, and we are projecting that it will work well in other databases. Future work includes finding new weighting schemes to improve the NLA architecture even more, find other ways to incorporate different nonlinear dimensionality techniques, and improve run times for a robust recognition algorithm.

REFERENCES

- [1] Svozil, D., Kvasnicka, V., and Pospichal, J., "Introduction to multi-layer feed-forward neural networks," *Chemometrics and intelligent laboratory systems* **39**(1), 43–62 (1997).
- [2] Morris, R. J. and Rubin, L. D., "Technique for object orientation detection using a feed-forward neural network," (Oct. 22 1991). US Patent 5,060,276.
- [3] Hornik, K., Stinchcombe, M., and White, H., "Multilayer feedforward networks are universal approximators," *Neural networks* **2**(5), 359–366 (1989).
- [4] Dony, R. D. and Haykin, S., "Neural network approaches to image compression," *Proceedings of the IEEE* **83**(2), 288–303 (1995).
- [5] Connor, J. T., Martin, R. D., and Atlas, L. E., "Recurrent neural networks and robust time series prediction," *Neural Networks, IEEE Transactions on* **5**(2), 240–254 (1994).
- [6] Liang, X.-B. and Wang, J., "A recurrent neural network for nonlinear optimization with a continuously differentiable objective function and bound constraints," *Neural Networks, IEEE Transactions on* **11**(6), 1251–1262 (2000).
- [7] Jim, K.-C., Giles, C. L., and Horne, B. G., "An analysis of noise in recurrent neural networks: convergence and generalization," *Neural Networks, IEEE Transactions on* **7**(6), 1424–1438 (1996).
- [8] Hopfield, J. J., "Neural networks and physical systems with emergent collective computational abilities," *Proceedings of the national academy of sciences* **79**(8), 2554–2558 (1982).
- [9] Seow, M.-J. and Asari, V. K., "Recurrent network as a nonlinear line attractor for skin color association," in *Advances in Neural Networks–ISNN 2004*, 870–875, Springer (2004).
- [10] Seow, M.-J. and Asari, V. K., "Recurrent neural network as a linear attractor for pattern association," *Neural Networks, IEEE Transactions on* **17**(1), 246–250 (2006).
- [11] Seow, M.-J., Alex, A. T., and Asari, V. K., "Learning embedded lines of attraction by self organization for pose and expression invariant face recognition," *Optical Engineering* **51**(10), 107201–1 (2012).
- [12] Happel, B. L. and Murre, J. M., "Design and evolution of modular neural network architectures," *Neural networks* **7**(6), 985–1004 (1994).
- [13] Gomi, H. and Kawato, M., "Recognition of manipulated objects by motor learning with modular architecture networks," *Neural Networks* **6**(4), 485–497 (1993).