Vector Quantization on RGBD image using Simulated Grossberg Network and Modified ART2 Network

Shiyao Wang

COMP 6759, Artificial Neural Network

Professor Siwei Lu.

December 14, 2015,

Contents

Vector Quantization on RGBD image using Simulated Grossberg Network and Modified ART2 Network

Shiyao Wang

Abstract

Using simulated Grossberg Network together with modified ART2 to perform vector quantization(VQ) on RGBD images is mainly discussed in this paper.

1. Problem Statement

Recently, RGBD textures[1] are increasingly found due to the development of depth-sensing cameras, so it is vital to apply image processing methodology upon RGBD images for specific application, e.g., vector quantization.

Vector quantization (VQ) is a classical quantization technique from data processing which was originally used for data compression. In this project, ART2[2] is chosen to perform this task.

Before that, normally, in depth images processing, we always focused on specific depth range, such as foreground or background. So, we should able to clip pixels we are not interested out of processing. After that, processing methodology for normal RGB textures can be applied. In order to obtain pixels within particular depth range, we need a method for data normalization, contrast enhancement and noise suppression, e.g., Grossberg Neural Network.

Grossberg network are initially built based on some electronic building block, i.e., differentiator and integrator circuits. But with fixed building structure, the function of the networks is strictly limited. So simulated implementation for these neural networks based on scientific computing method would be discussed in this paper [1].

[1] Normal RGB texture with depth

[2] Adaptive Resonance Theory 2

In this paper, depths matrix (Figure 1) capture by depth-sensing camera would be sent to the simulated Grossberg Net by column vetors. And then processing result would be use in the alpha channel to perform texture clipping. In the end, the result image pixels would be passed to modified ART2 as the input using their RGB color channel, and also alpha channel would be used for identify the corresponding cluster.

0	0	24472	24264	24472	24264	24472	24264	24264	24472	..	17064	17064
0	0	24472	24472	24472	24264	24264	24048	24264	24264	..	17064	17064
0	0	24472	24264	24264	24264	24264	24264	24264	24472	..	17064	17064
0	0	24264	24048	24048	24048	24048	24048	24264	24264	..	17176	17176
0	0	24048	24048	24048	24048	24048	24048	24264	24264	..	17176	17064
0	0	23640	23840	23840	23840	24048	24048	24048	23840	..	17176	17064
0	0	23640	23840	23840	23840	24048	24048	24048	24048	..	17176	17176
0	0	23640	23840	23640	23840	24048	23840	24048	23840	..	17176	17176
0	0	23640	23640	23640	23840	23840	23640	23840	23840	..	17176	17064
0	0	23640	23640	23640	23840	23840	23640	23840	23840	..	17176	17176
0	0	23640	23640	23640	23840	23840	23640	23840	24048	..	17176	17064
0	0	23440	23440	23640	23840	23840	23640	24048	24048	..	11968	11968
0	0	23640	23640	23840	23840	23840	23640	23840	24048	..	11912	11912
0	0	23440	23640	23640	23840	23840	23640	23840	24048	..	11912	11912
..	..	..	..	..	..	..	..	..	..	..	..	..
0	0	21112	21112	21112	21112	21112	21272	21112	21112	..	11376	11376
0	0	21112	21112	21112	21272	21272	21112	21112	21112	..	11376	11376

Figure 1. RGBD Texture Sample: normal RGB texture on the left, and depths matrix on the right

2. Introduction

2.1. Vector Quantization

Vector Quantization [2] [3] [4] works by dividing a large set of data (vectors) into groups having approximately the same value. Each group is represented by its centroid point, as in k-means and some other clustering algorithms.

Vector quantization is based on the competitive learning paradigm, so it is closely related to the artificial neural networks which can perform clustering based on competitive layers [5].

2.2. The Grossberg Network

The Grossberg network is introduced by Stephen Grossberg. It is heavily influenced by the human visual system, which is a self-organizing continuous-time competitive network. [6].

The first layer of the Grossberg network represents the retina, which normalizes the input pattern. It illustrates how the visual system combine on-center/off-surround connection patterns and a shunting model to perform an automatic gain control, which normalizes total activity.

The second layer of the Grossberg network is a rough model of the operation of the visual cortex performs a

competition, which contrast enhances the output pattern and stores it in short-term memory. Nonlinear feedback and the on-center/off-surround connection pattern is also used to produce the competition and the storage. The choice of the transfer function and the feedback connection pattern determines the degree of competition (e.g., winner-take-all, mild contrast enhancement, or no change in the pattern).

2.2.1. Basic Sturcture of Grossberg Network.

Grossberg Network was inspired by the operation of the mammalian visual system, which contains three components: Layer 1, Layer 2 and the adaptive weights. The network includes *short-term memory* (**STM**) and *long-term memory* (**LTM**) mechanisms, and performs normalization and contrast enhancement.(Figure 2)

2.2.2. Layer 1 of Grossberg Network.

Layer 1 of the Grossberg network receives external inputs and normalizes the intensity of the input pattern. And it is based on shunting model.

The equation of operation of Layer 1 is:

$$\varepsilon \frac{\mathrm{d}n_1(t)}{\mathrm{d}t} = -n_1(t) + (b_1^+ - n_1(t))[W_1^+]p - (n_1(t) + b_1^-)[W_1^-]p \tag{1}$$

where b_1^+ is upper bound and b_1^- as the lower bound of the response $n_1(t)$, and W_1^+ as well as W_1^- perform the **On-Center/Off-Surround Connection** of input parttern p as shown in Figure 3.

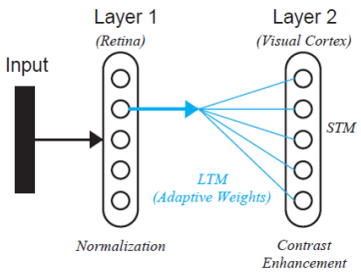


Figure 2. Grossberg Network

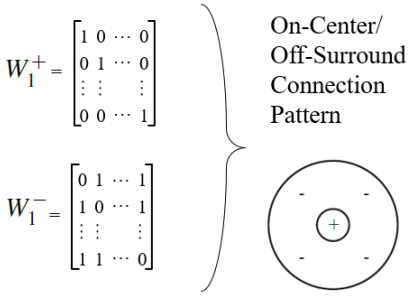


Figure 3. On-Center/Off-Surround Connection

2.2.3. Layer 2.

Layer 2 of the Grossberg network, performs several functions. First, it contrast enhances its pattern, so that the neuron that receives the largest input will dominate the response. Second, it operates as a short-term memory (STM) by storing the contrast-enhanced pattern. As with Layer 1, the shunting model forms the basis for Layer 2. The main difference between Layer 2 and Layer 1 is that Layer 2 uses feedback connections. The feedback enables the network to store a pattern, even after the input has been removed. The feedback also performs the competition that causes the contrast enhancement of the pattern. The equation of operation of Layer 2 is:

$$\varepsilon \frac{\mathrm{d}n_2(t)}{\mathrm{d}t} = -n_2(t) + (b_2^+ - n_2(t))\{[W_2^+]f(n_2(t)) + \mathbf{Wa_1}\} - (n_2(t) + b_2^-)[W_2^-]f(n_2(t)) \tag{2}$$

The W_2^+ and W_2^- are same as Layer 1, performs on-center / off-surround connections. b_2^+ and b_2^- are upper / lower bounds of the response. $f(n_2(t))$ is called **transfer function**, W is the **adaptive weights** and a_1 would be the input from Layer 1.

2.2.4. Adaptive Weights and Learning Rule.

The learning law for the adaptive weights is important in Grossberg Net. These adaptive weights is called the **long-term memory** (LTM). This is because the rows of will represent patterns that have been stored and that the network will be able to recognize:

$$\frac{\mathrm{d}w_{i,j}(t)}{\mathrm{d}t} = \alpha n_{2-i}(t)\{-w_{i,j}(t) + n_{1-j}(t)\} \tag{3}$$

where $n_{2-i}(t)$ is the i-th output of Layer 2, $n_{1-j}(t)$ is the j-th output of Layer 1 and α is the called learning rate.

We can see that if n_{2-i} is nonzero, i.e., active, then the output of Layer 1 would be store at the i-th row of adaptive weights. Therefore, when a similar vector is input to Layer 2, since the adaptive weights W is having the inner product with it, then it would make i-th neuron to become bigger. Layer 2 then performs a competition between the neurons, which tends to **contrast enhance** the output pattern, maintaining large outputs while attenuating small outputs.

2.2.5. Transfer Function.

The transfer function $f(n_2(t))$ have a great impact on the behavior of Layer 2. Suppose that an input has been applied for some length of time that stabilized to some pattern. If the input is then removed, Figure 7 illustrates the responce of Layer 2 on a transfer function faster than linear performs a winner-take-all competition(Input is moved when $t = 0.25$). And Figure 4 demonstrates how the $f(n_2(t))$ will affect the steady state response of the network.

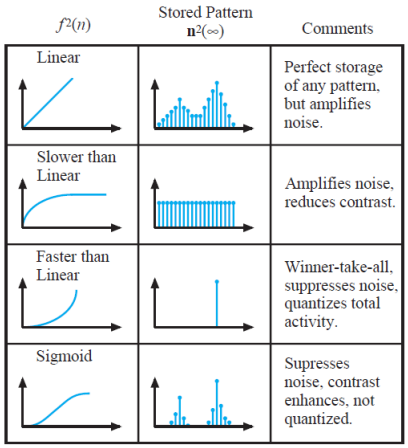


Figure 4. Choice of Transfer Function

2.3. Adaptive Resonance Theory 2

Grossberg showed that the standard competitive networks do not have stable learning in response to arbitrary input patterns. This is the key problem of the competitive network, as it does not always form stable clusters (or categories). The network's adaptability (or plasticity) lead to the learning instability, which causes prior learning to be eroded by more recent learning. Grossberg refers this problem as the "stability/plasticity dilemma.".

In order to address the stability/plasticity dilemma, Grossberg and Gail Carpenter developed a theory, called adaptive resonance theory (ART) [7]. It is based on the Grossberg network. The key innovation of ART is the use of "expectations". When an input pattern is presented to the network, it is compared with the prototype vector store in the memory (adaptive weights) that it most closely matches (the expectation). If the match between the prototype and the input vector is not adequate, a new prototype (cluster) is selected. In this way, previously learned memories

(prototypes) would not be eroded by new learning.

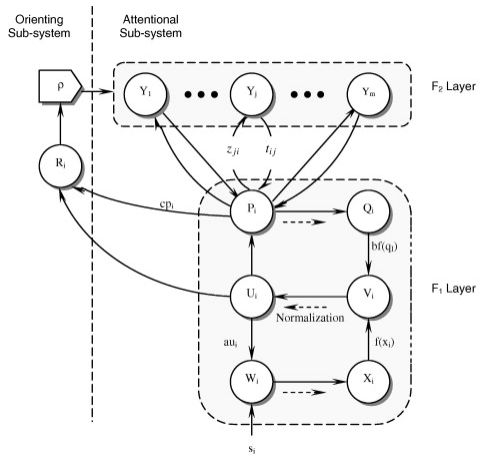
Carpenter and Grossberg developed a variation of ART1, called ART2 [8], to handle analog patterns instead of binary ones. In ART2, several sublayers take the place of Layer 1. while the other structure of ART2 remain very similar to ART1. Unlike binary vectors, analog vectors can be arbitrarily close together, so sublayers in Layer 1 are needed for dealing with this situation.

ART2 is designed for clustering continuous-valued vectors. Input patterns may be presented to the net in any arbitrary order. Each time a pattern is presented, an appropriate cluster unit is chosen and that cluster's weights are adjusted to let the cluster unit learn the pattern.

2.3.1. Normal Stucture of ART2.

Figure 5 is the basic structure of ART2, which accepts analog signal. It is an unsupervised neural network can perform clustering automatically.

Unit activations are described as follows:



$$
\begin{aligned}
w_i &= s_i + a u_i \\
x_i &= \frac{w_i}{e + ||w||} \\
v_i &= f(x_i) + b f(q_i) \\
u_i &= \frac{v_i}{e + ||v||} \\
p_i &= u_i + \sum_j g(y_j) t_{ij} \\
q_i &= \frac{p_i}{e + ||p||}
\end{aligned}
$$

Figure 5. ART 2 Architecture

The activation function

$$
f(x) = \begin{cases} 0 & 0 \le x \le \theta \\ x & x > \theta \end{cases}
$$

$$g(y_j) = \begin{cases} d & j = max\{\sum_{i=1}^{n} p_j z_{ji} | j = 1,2,\ldots,m\} \\ 0 & \text{otherwise} \end{cases}$$

z_{ji} is bottom-up weights between F_1 Layer and F_2 Layer, while t_{ij} is top-down weights.

The F_2 layer is a competitive layer: The cluster unit (y) with the largest net input becomes the candidate to learn the input pattern. The activations of all other F_2 units are set to zero. If the cluster unit is not allowed to learn, it is inhibited and a new cluster unit is selected as the candidate. The degree of similarity required for patterns to be assigned to the same cluster unit is controlled by a user-specified parameter, known as the vigilance parameter. For investigating the reset condition, we updated u_i, p_i and calculated r_i as follow:

$$r_i = \frac{u_i + cp_i}{||u|| + c||p||}$$

And if $||r|| < \rho$, it means that u and p are not parallel enough to pass the vigilance test, the cluster unit is not allowed to learn, it is inhibited and test for the new cluster until it passes the vigilance test. If it finally past the vigilance test, it is called "**Resonance**". After resonance, the corresponding row of top-down weights and the column of bottom-up weights are updated as follows:

$$t_{Ij} = z_{jI} = \frac{u_j}{1-d} \quad \text{(I is the cluster number)}$$

3. Previous Work

Due to the unstability of Grossberg network, it is treated as an important basic model and concept in neural network fields. ART2 is an advanced model of Grossberg network, and is widely used for image processing including image segamentation [9] [10] [11] and Vector Quantization [12] [13]. However, there is still no research or works using these neural networks on vector quantization of RGBD images.

4. Project

4.1. Simulated Grossberg Net

From Eq 1, Eq 2 and Eq 3, the problem are all related to solving ordinary differential equations(ODEs) with a given initial value, known as initial value problem. Therefore, initial value problem turns out to be the basic problem.

In this paper, we discussed Heun's method [14] [15] [16]. Heun's method also known as Preditor-Corrector method, is also called the improved or modified Euler's method, or a similar two-stage RungeKutta method. It is named after Karl Heun and is a numerical procedure for solving the initial value problem.

4.2. Methods on Initial Value Problem

Given the initial value problem:

$$\begin{aligned} y^{(1)}(t) &= f(t, y(t)) \\ y(t_0) &= y_0 \end{aligned}$$

- We know the state y_0 of a system at time t_0.
- We understand how the system evolves (through the ordinary differential equation).
- We want to approximate the state in the future $[y(t_0 + h)]$.

4.2.1. Euler's Method.

Now we want to approximate the solution at $t_0 + h$; therefore, according to Taylor's therom:

$$y(t_0 + h) = y(t_0) + y^{(1)}(t_0)h + \frac{1}{2}y^{(2)}(\tau)h^2 \tag{4}$$

where $\tau \in (t_0, t_0 + h)$

if we replace the initial condition and derivative from the ODE, we can get:

$$y(t_0 + h) \approx y_0 + h \cdot f(t_0, y_0)$$

with a truncation error $\frac{1}{2}y^{(2)}(\tau)h^2$

4.2.2. Heun's Method.

For Euler's Method, the approximations simply use the value at one end-point to approximating the integral of a function over an interval. This is a simplest approximations for the integral value.

We could get a better result by approximating the integral using the average of the two end-points:

$$\int_a^b g(x)\mathrm{d}x \approx \frac{g(a)+g(b)}{2}(b-a)$$

This is the trapezoidal rule of integration.

We would try to apply trapezoidal rule to improve Euler's Method, which is Heun's Method. The problem is, we would have to know the slope at t_0+h in order to approximate. Note, however, that Euler's method gives us an approximation of $y(t_0+h) \approx y_0 + h \cdot f(t_0, y_0)$. Therefore, we can approximate the slope at t_0+h with:

$$y^{(1)}(t_0+h) = f(t_0+h, y(t_0+h)) \approx f(t_0+h, y_0+h\cdot f(t_0,y_0))$$

Applying the same principle as the trapezoidal rule, we would then approximate:

$$y(t_0+h) \quad \approx \quad y_0 + h\frac{f(t_0,y(t_0))+f(t_0+h, y_0+h\cdot f(t_0,y_0))}{2}$$

4.3. Problems

Simulated Grossberg Network using Heun's method would face some problems:

1. Hard to choose step h.
2. Slow speed of converge due to massive calculation of matrix W^+ and W^-.

Due to these factors, we should find another way for simulation.

4.4. Observation

To investigate the normalization effect of Layer 1, just assume that b_1^- is zero, and $b_1^+ = \{b^+, b^+, \ldots, b^+\}$, consider the response of neuron i:

$$\varepsilon \frac{\mathrm{d}n_i(t)}{\mathrm{d}t} = -n_i(t) + (b^+ - n_i(t))p_i - n_i(t)\sum_{j \neq i} p_j$$

In the steady state, we have

$$(1 + \sum_{j=1} p_j) n_i(t) = b^+ p_i$$

Define relative intensity of node i to be $\tilde{p}_i = \frac{p_i}{S}$ where $S = \sum_{j=1} P_j$, then we would have

$$n_i(t) = (\frac{b^+ S}{1+S})\tilde{p}_i$$

Therefore, $n_i(t)$ will be proportional to the relative intensity $\tilde{p}_i$. In addition, the total neuron activity is bounded by b^+.

And similarly, if we analyze the response of Layer 2 in steady state, we would get:

$$n_2(t)[1 + \sum f(n_2(t))] = f(n_2(t)) + \mathbf{Wa_1}$$

The learning rule for adaptive weights is instar rule, quick learning strategy can be easily adapt to it.

The response of Layer 1 is similar like shunting model, which is shown in Figure 6. And the response of Layer 2 is illustrates in Figure 7.

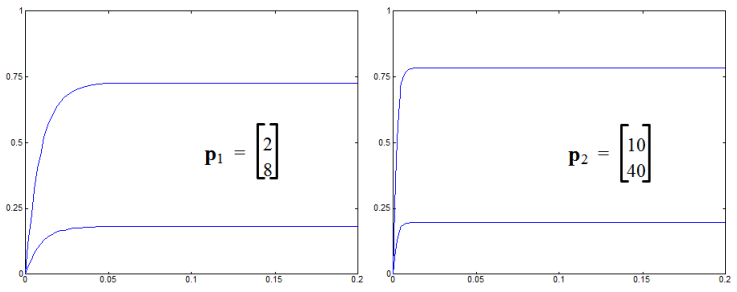


Figure 6. Layer 1 Response Example

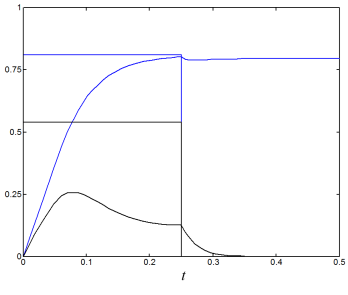


Figure 7. Layer 2 Response Example

4.4.1. Quicker Converge.

Use the investigation result of converge state, some modification is used to upgrade the simulation:

- As W^+ is identity matrix, ignore the calculation related to it.
- As W^- perform an off-surround effect, instead of calculating inner product, we calculate the sum of total input pattern, and subtract every element to perform an off-surround effect.
- Perform quick converge calculation on Layer 1(upper bound=1, lower bound=0): $output = \frac{input}{\sum input+1}$.
- Perform quick converge calculation on Layer 2(upper bound=1, lower bound=0): $output_i = \frac{f_i(output)+[Wa]_i}{\sum_1^n f_i(output)+[Wa]_i+1}$
- Perform quick learning: $\{W_i = a_i | output_i = max_1^n(output)\}$

4.5. Grossberg Net Processing Result

Input column vectors of the depths matrix, and adapt the output into the alpha channel, here is the result:

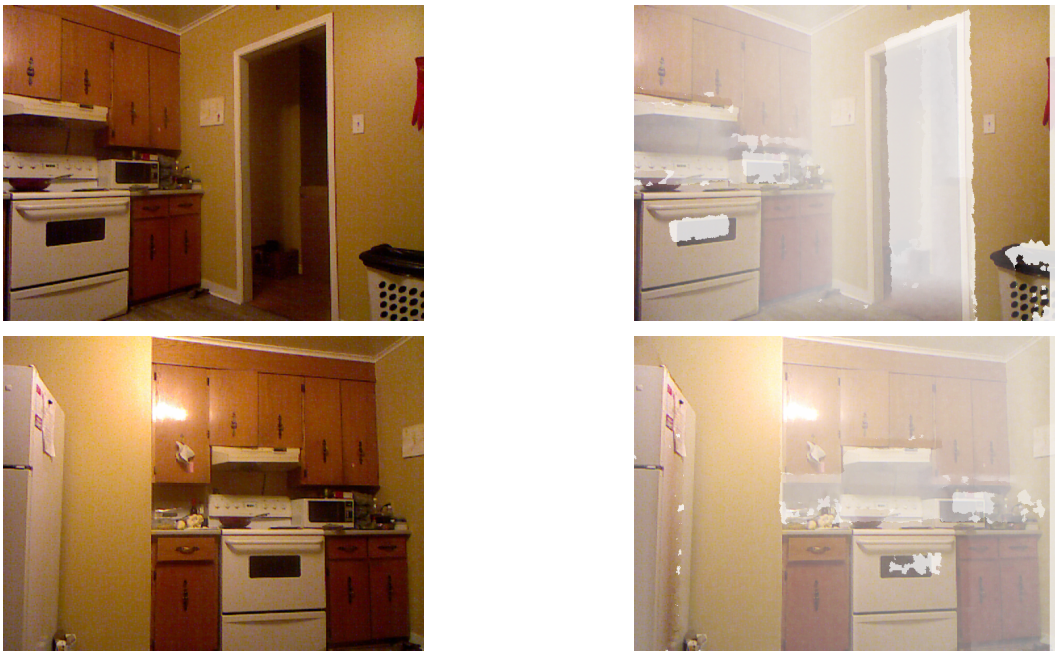


Figure 8. (a)(c) Image of my kitchen, (b)(d) Process after Grossberg Net (Target depth = 1.5 meter)

4.6. Simulated ART2

4.6.1. Some problems of normal ART2.

The normal structure of ART2 has several problems:

1. The output size, which is also the cluster number, is previously decide.
2. The operation of the net depend much on the parameter which is hard to decide.
3. The top-down and bottom-up weights are the same after quick learning.
4. The magnitude of the input pattern are not stored, as both weights is updated according to u which $||u|| = 1$.
5. Almost every cluster should be test before it coubld be cluster to a fresh new cluster.
6. To wait the equilibrium of F_1 Layer could take very long time, if the input is just in the middle of two existed cluster.
7. The function use inner product to choose the biggest element for the vector which is most close to the corresponding cluster.
8. For quick learning, the network can only remember the more recent input.

Some of the problems really lead to big trouble for processing the vector quantization, as we would not able to figure out how many clusters is enough. Therefore it is significant to modify classical ART2 to make it suitable for our task.

4.6.2. Modified Structure of ART2.

Modified ART2 [9] has following modifications:

1. Adapting minimum-win competitive rule in activation mechanism of F_2 nodes based on Euclidian-distance.
2. Top-down and bottom-up weights are combined into just one weight matrix W.
3. The size of output layer is dynamically, and all new nodes are added when needed.

4. The activation function of p is changed to:

$$p_i = h_i + \sum_j g(y_j) W_{ij}, \text{ and } h_i = ||S|| \cdot u_i$$

where S is the input pattern.

5. Introducing three special functions to assistant similarity measurement between input pattern and stored pattern:

$$H = ||X|| \cdot U$$

$$L = \begin{cases} W_I' & if ||W_I'|| \neq 0 \\ H & if ||W_I'|| = 0 \end{cases}$$

$$R = exp(\frac{-||H-L||}{\delta})$$

6. Weight update using algorithm similar with K-means:

$$W_{new} = \frac{1}{k}((k-1)W_{old} + input)$$

which k refers to the activation times of this node.

7. Perform quick searching among F_1 Layer, It means that when the node with the minimum Euclidian-distance among all learned nodes don't pass vigilance-testing then the others will also not pass. Therefore directly create a new node will greatly shorten searching time.

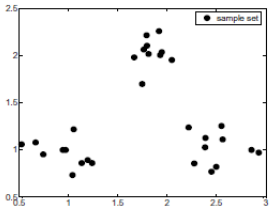
8. Ignore pixels with smaller alpha channel to adapt the input from Grossberg Network.

4.6.3. Experimental Results and Analysis.

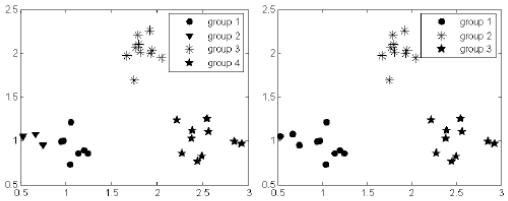
A clustering experiment between normal and modified ART2 is being processed, the result is shown in Fig 9. We can see that the modified ART2 have better clustering result.

4.7. ART2 Processing Result

Some VQ result based on modified ART2 for Lenna picture are shown in Fig 10, we choose different vigilence parameter (ρ) for the processing, and other parameters remain the same ($a = b = 5, e = \theta = 0, \delta = 1.2247$)

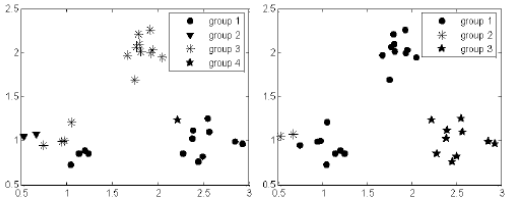


(a) samples



(b) $\rho = 0.9, 0.85$ from left to right

(b) Result by modified ART2



(c) $\rho = 0.996, 0.99$ from left to right

(c) Result by original ART2

Figure 9. Experiment on normal ART2 and modified ART2

(a) Original (b) $\rho = 0.7788$ (c) $\rho = 0.85$ (d) $\rho = 0.92$

Figure 10. Process Result of Lenna Image for Modified ART2

Some statistics of processing is shown in Table 1.

ρ	Number of Cluster	Processing Time(ms)
0.7788	9	87
0.85	20	519
0.92	97	707

Table 1. Analyze of Processing Result

5. Results

First, input the column vectors of the depths matrix of the RGBD image into the simulated Grossberg Net, and use the output as the alpha channel. Then use the RGB channel for every pixel as a 3-dimentional vector, and use alpha channel as the noise, input these data pixel by pixel into the modified ART2.

At last, here are the final processing result of my project:

Figure 11. Processing by Grossberg Network and Modified ART2

6. Conclusion

The simulated Grossberg Network is suitable for solving normalization and contrast enhancement problem. However, the processing time for Grossberg Net to converge is quite long, as the net perform an off-surround calculation relate to every nodes in the network. As the size of network grows, the calculation time and converge time will increase dramatically. So in this paper, a quicker convergence modification is used for processing.

The original ART2 is not good enough to perform color clustering due to its limitations. But modified ART2 perform well in clustering not only the better processing result, but also less time consuming. The vigilence parameter ρ is the key parameter to control the final cluster number, which should be choosed wisely, as it also have significant impact on the total processing time.

7. Future Work

Vector quantization is a preprocessing step for so many image processing algorithm. Adapt the processing result for further image processing such as image segamentation and gesture, human and skeleton recognition. In addition, modified ART2 can be implemented for other clustering problems.

References

[1] U. M. Ascher and L. R. Petzold, *Computer methods for ordinary differential equations and differential-algebraic equations*. Siam, 1998, vol. 61.

[2] A. Gersho and R. M. Gray, *Vector quantization and signal compression*. Springer Science & Business Media, 2012, vol. 159.

[3] N. M. Nasrabadi, R. King *et al.*, "Image coding using vector quantization: A review," *Communications, IEEE Transactions on*, vol. 36, no. 8, pp. 957–971, 1988.

[4] R. M. Gray, "Vector quantization," *ASSP Magazine, IEEE*, vol. 1, no. 2, pp. 4–29, 1984.

[5] S. C. Ahalt, A. K. Krishnamurthy, P. Chen, and D. E. Melton, "Competitive learning algorithms for vector quantization," *Neural networks*, vol. 3, no. 3, pp. 277–290, 1990.

[6] S. Grossberg, E. Mingolla, and D. Todorovic, "A neural network architecture for preattentive vision," *Biomedical Engineering, IEEE Transactions on*, vol. 36, no. 1, pp. 65–84, 1989.

[7] G. A. Carpenter and S. Grossberg, "A massively parallel architecture for a self-organizing neural pattern recognition machine," *Computer vision, graphics, and image processing*, vol. 37, no. 1, pp. 54–115, 1987.

[8] ——, "Art 2: Self-organization of stable category recognition codes for analog input patterns," *Applied optics*, vol. 26, no. 23, pp. 4919–4930, 1987.

[9] J. Ai, B. Funt, and L. Shi, "A new type of art2 architecture and application to color image segmentation," in *Artificial Neural Networks-ICANN 2008*. Springer, 2008, pp. 89–98.

[10] L. Cinque, G. Foresti, and L. Lombardi, "A clustering fuzzy approach for image segmentation," *Pattern Recognition*, vol. 37, no. 9, pp. 1797–1807, 2004.

[11] N. Yeo, K. Lee, Y. Venkatesh, and S. H. Ong, "Colour image segmentation using the self-organizing map and adaptive resonance theory," *Image and Vision Computing*, vol. 23, no. 12, pp. 1060–1079, 2005.

[12] M. Dabestani and M.-S. Moin, "A fast vq codebook generation algorithm using art2 neural network."

[13] N. Vlajic and H. C. Card, "Vector quantization of images using modified adaptive resonance algorithm for hierarchical clustering," *Neural Networks, IEEE Transactions on*, vol. 12, no. 5, pp. 1147–1162, 2001.

[14] K. E. Atkinson, *An introduction to numerical analysis*. John Wiley & Sons, 2008.

[15] W. Chen, D. de Kee Fong, and P. N. Kaloni, *Advanced mathematics for engineering and science*. World Scientific, 2003.

[16] J. J. Leader, *Numerical analysis and scientific computation*. Pearson Addison Wesley Boston, 2004.

[17] M. T. Hagan, H. B. Demuth, M. H. Beale *et al.*, *Neural network design*. Pws Pub. Boston, 1996.

[18] K. Lankalapalli, S. Chatterjee, and T. Chang, "Feature recognition using art2: a self-organizing neural network," *Journal of Intelligent Manufacturing*, vol. 8, no. 3, pp. 203–214, 1997.

[19] S. Grossberg, "Adaptive pattern classification and universal recoding: I. parallel development and coding of neural feature detectors," *Biological cybernetics*, vol. 23, no. 3, pp. 121–134, 1976.

[20] ——, *Studies of mind and brain: Neural principles of learning, perception, development, cognition, and motor control*. Springer Science & Business Media, 2012, vol. 70.

[21] ——, *Studies of Mind and Brain: Neural Principles of Learning, Perception, Development*. Cognition, and Motor Control. Reidel Publishing Co., Boston, 1982.

Appendix

A. Building Blocks of Grossberg Network

Here are some building blocks of the Grossberg network.

A.1. Leaky Integrator

The basic building block is the "leaky" integrator, which is shown in Figure 12. The equation for it is:

$$\varepsilon \frac{\mathrm{d}n(t)}{\mathrm{d}t} = -n(t) + p(t) \tag{5}$$

where ε is the system *time constant*.

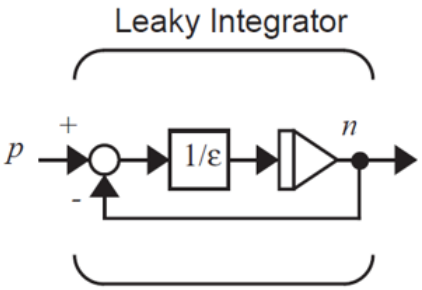


Figure 12. Leaky Integrator

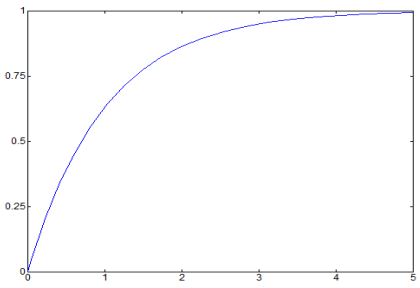


Figure 13. Leaky Integrator Response

A graph of its response, for $p = 1$ and $\varepsilon = 1$, is given in Figure 13. The response exponentially approaches a steady state value of 1.

There are two important properties of the leaky integrator. First, because Eq.(5) is linear, if the input is scaled, then the response will be scaled by the same amount. For example, if the input is doubled, then the response will also be doubled, but will maintain the same shape. Second, the speed of response of the leaky integrator is determined by the time constant ε. When ε decreases, the response becomes faster; when ε increases, the response becomes slower.

A.2. Shunting Model

The leaky integrator forms one of Grossberg's fundamental neural models: the shunting model, which is shown in Figure 14.

The equation of operation of this network is:

$$\varepsilon \frac{dn(t)}{dt} = -n(t) + (b^+ - n(t))p^+ - (n(t) + b^-)p^- \tag{6}$$

where p^+ is nonnegative value representing the **excitatory** input to the network, and p^- is a nonnegative value representing the **inhibitory** input. The biases b^+ and b^- are nonnegative constants that determine the upper and lower limits on the response.

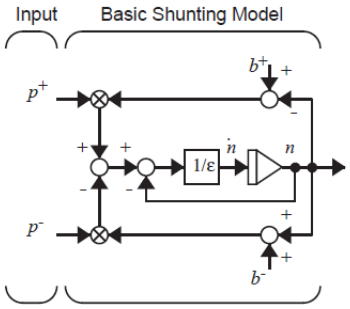


Figure 14. Shunting Model

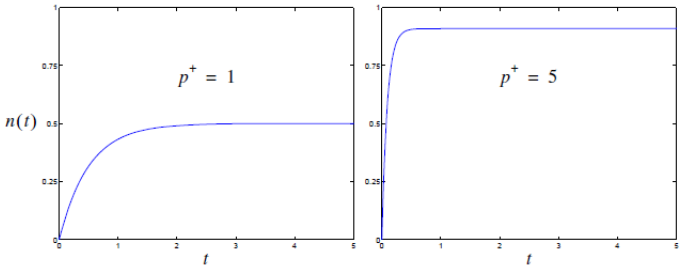


Figure 15. Shunting Model Response

If we analyze the stable state of the shunting model, which $\frac{dn(t)}{dt} = 0$, we can get

$$-n(t) + (b^+ - n(t))p^+ - (n(t) + b^-)p^- = 0$$

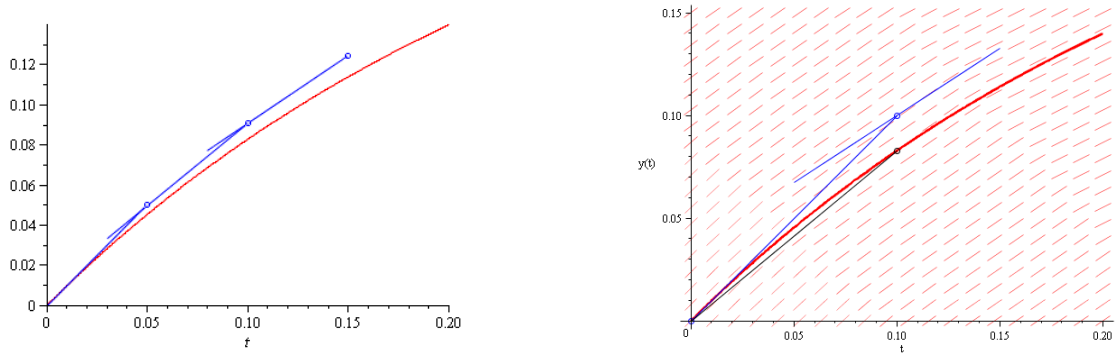
If we apply any exictatory input p^+, we will get $n(t) = \frac{p^+}{1+p^+}b^+$, as p^+ is nonnegative value, no matter how big the input p^+ is, the response $n(t)$ has upper bound b^+. Similarly, if we apply an inhibitory input p^-, we would get the same result that the response $n(t)$ has lower bound $-b^-$. Figure 15 illustrates the performance of the shunting network when $b^+ = 1$, $b^- = 0$ and $\varepsilon = 1$.

B. Euler's method and Heun's method

Here are some figures actually show how Euler's method and Heun's method works:

(a) Euler's method (b) Heun's method

Figure 16. How to approximate integral value



(a) Euler's method (b) Heun's method

Figure 17. How to solve Initial Value Problem

C. Code Samples

C.1. Grossberg Network

```
\\Layer 1
private Vector<float> Layer1Function(float t, Vector<float> nt, object sum)
{
    float _sum = (float)sum;
    Vector<float> result = Vector<float>.Build.Dense(nt.Count, 0f);
    for (int i = 0; i < nt.Count; i++)
    {
        result[i] = -nt[i] + ((_uplimit - nt[i]) * _input[i])
                    - ((nt[i] + _lowlimit) * (_sum-_input[i]));
    }
```

```
    return result / _epsilon;
}

private void NextStep()
{
    Vector<float> measure = Layer1Function(_time, _output, _inputSum);
    if (measure.L1Norm() < System.Math.Pow(10, -5))
    {
        return;
    }
    float h = 0.1f * _epsilon * (_uplimit - _lowlimit)
                / System.Math.Max((float)_input.L1Norm(), (_uplimit - _lowlimit));
    float nextTime = _time + h * 100;
    _output = DifferentialEquation.HeunMethod(_time,
                              _output, nextTime, h, Layer1Function, _inputSum);
    _time = nextTime;
}
\\Layer 2
private Vector<float> Transfer(Vector<float> n)
{
    Vector<float> result = n;
    float divider = (_uplimit - _lowlimit) / 10;
    float midpoint = (_uplimit + _lowlimit) / 2;
    for (int i = 0; i < n.Count; i++)
    {
        float p = (n[i] - midpoint) / divider;
        result[i] = 1 / (1 + Mathf.Exp(-p));
    }
    return 50f * (_uplimit - _lowlimit) * result + _lowlimit;
}

private Vector<float> Layer2Function(float t, Vector<float> u, object parameter)
{
    Vector<float> sig = Transfer(u);
    float _sum = sig.Sum();
    Vector<float> result = Vector<float>.Build.Dense(u.Count, 0f);
    for (int i = 0; i < u.Count; i++)
    {
        result[i] = -u[i] + ((_uplimit - u[i]) * (sig[i] + feedback[i]))
                      - ((u[i] + _lowlimit) * (_sum - sig[i]));
    }
    return result / _epsilon;
}

private Vector<float> InstarFunction(float t, Vector<float> u, object ni)
{
    return _learningRate * (float)ni * (_layer1.Output - u);
}

private void InstarLearning(float time, float nextTime, float h)
{
    for (int i = 0; i < _adaptiveWeights.RowCount; i++)
    {
```

```
        Vector<float> newRow = DifferentialEquation.HeunMethod(time,
      _adaptiveWeights.Row(i), nextTime, h, InstarFunction, _layer1.Output[i]);
        _adaptiveWeights.SetRow(i, newRow.ToArray());
    }
}

private void NextStep()
{
    Vector<float> measure = Layer2Function(_time, _output, null);
    if (measure.L1Norm() < System.Math.Pow(10, -5))
    {
        return;
    }
    float h = 0.1f * _epsilon;
    float nextTime = _time + h * 100;
    _output = DifferentialEquation.HeunMethod(_time,
             _output, nextTime, h, Layer2Function, null);
    InstarLearning(_time, nextTime,h);
    _time = nextTime;
}
```

C.2. Modified Grossberg Network

```
\\Layer 1
private void NextStep()
{
    for (int i = 0; i < _output.Count; i++)
    {
        _output[i] = _input[i] * _uplimit / (1 + _inputSum);
    }
}
\\Layer 2
private Vector<float> Transfer(Vector<float> n)
{
    Vector<float> result = n;
    float divider = (_uplimit - _lowlimit) / 10;
    float midpoint = (_uplimit + _lowlimit) / 2;
    for (int i = 0; i < n.Count; i++)
    {
        float p = (n[i] - midpoint) / divider;
        result[i] = 1 / (1 + Mathf.Exp(-p));
    }
    return 50f * (_uplimit - _lowlimit) * result + _lowlimit;
}

private void InstarLearning()
{
    _adaptiveWeights.SetRow(_output.MaximumIndex(), _layer1.Output);
}

private void NextStep()
{
    for (int j = 0; j < 50; j++)
```

```
    {
        Vector<float> trans = Transfer(_output);
        float _sum = trans.Sum();
        for (int i = 0; i < _output.Count; i++)
        {
            _output[i] = (trans[i] + feedback[i]) / (_sum + feedback[i] + 1);
        }
    }
    InstarLearning();
}
```

C.3. Modified ART2

```
\\Layer 1
....
public class Orienting
{
    ....
    public bool Reset(params Vector<float>[] inputs)
    {
        Vector<float> _inputLayer = inputs[0];
        Vector<float> _uLayer = inputs[1];
        Vector<float> _weights = inputs[2];
        Vector<float> H = (float)_inputLayer.L2Norm() * _uLayer;

        Vector<float> L = H;

        if ((float)_weights.L2Norm() > float.Epsilon)
        {
            L = _weights;
        }

        Vector<float> diff = H - L;

        float measure = (float)diff.L2Norm() / _delta;

        return System.Math.Exp(-measure) < _p;
    }
    ....
}
public abstract class NormalizedLayer : Layer
{
    ....
    public bool Input(params Vector<float>[] inputs)
    {
        if (_theLayer == null)
        {
            _theLayer = Vector<float>.Build.Dense(inputs[0].Count, 0f);
        }
        Vector<float> _inLayer = inputs[0];
        Vector<float> newVal = _inLayer / (_e + (float)_inLayer.L2Norm());
        bool converge = System.Math.Abs((_theLayer - newVal).L2Norm()) < _tol;
        _theLayer = newVal;
```

```
        return converge;
    }
    ....
}
public class XLayer : NormalizedLayer{}
public class ULayer : NormalizedLayer{}
public class QLayer : NormalizedLayer{}
public class WLayer : Layer
{
    ....
    public bool Input(params Vector<float>[] inputs)
    {
        if (_wLayer == null)
        {
            _wLayer = Vector<float>.Build.Dense(inputs[0].Count, 0f);
        }
        if (inputs[1] == null)
        {
            inputs[1] = Vector<float>.Build.Dense(inputs[0].Count, 0f);
        }
        Vector<float> newVal = inputs[0] + _a * inputs[1];
        bool converge = System.Math.Abs((_wLayer - newVal).L2Norm()) < _tol;
        _wLayer = newVal;
        return converge;
    }
    ....
}
public class VLayer : Layer
{
    ....
    public bool Input(params Vector<float>[] inputs)
    {
        if (_vLayer == null)
        {
            _vLayer = Vector<float>.Build.Dense(inputs[0].Count, 0f);
        }
        if (inputs[1] == null)
        {
            inputs[1] = Vector<float>.Build.Dense(inputs[0].Count, 0f);
        }
        Vector<float> newVal = function(inputs[0]) + _b * function(inputs[1]);
        bool converge = System.Math.Abs((_vLayer - newVal).L2Norm()) < _tol;
        _vLayer = newVal;
        return converge;
    }

    private Vector<float> function(Vector<float> _input)
    {
        Vector<float> result = Vector<float>.Build.Dense(_input.Count, 0f);
        for (int i = 0; i < _input.Count; i++)
        {
            if (_input[i] > _theta)
            {
```

```
                result[i] = _input[i];
            }
        }
        return result;
    }
    ....
}
public class PLayer : Layer
{
    ....
    public bool Input(params Vector<float>[] inputs)
    {
        if (_pLayer == null)
        {
            _pLayer = Vector<float>.Build.Dense(inputs[0].Count, 0f);
        }
        if (inputs[2] == null)
        {
            inputs[2] = Vector<float>.Build.Dense(inputs[0].Count, 0f);
        }
        Vector<float> newVal = (float)inputs[0].L2Norm() * inputs[1] + inputs[2];
        bool converge = System.Math.Abs((_pLayer - newVal).L2Norm()) < _tol;
        _pLayer = newVal;
        return converge;
    }
    ....
}
public void AddNode()
{
    _Weights = _Weights.Append(_pLayer.Output().ToColumnMatrix());
}

public void ModifiedLearning(Vector<float> input, int winner, int K)
{
    Vector<float> oldtdW = _Weights.Column(winner);
    Vector<float> newtdW = ((K - 1) * oldtdW + input) / K;
    _Weights.SetColumn(winner, newtdW.ToArray());
}
....
\\Layer 2
....
public bool Input(params Vector<float>[] inputs)
{
    _yLayer = inputs[0];
    bool convergence = (Winner() == lastWinner);
    lastWinner = Winner();
    return convergence;
}
public Vector<float> Output()
{
    if (_yLayer == null)
    {
        return null;
```

```
    }
    Vector<float> result = Vector<float>.Build.Dense(_yLayer.Count, 0f);
    if (Winner() >= 0)
    {
        result[Winner()] = 1;
    }
    return result;
}
public int Winner()
{
    int minIndex = -1;
    float min = float.PositiveInfinity;
    if (_yLayer != null)
    {
        for (int i = 0; i < _yLayer.Count; i++)
        {
            if (_yLayer[i] < min)
            {
                minIndex = i;
                min = _yLayer[i];
            }
        }
    }
    return minIndex;
}
....
\\Main Class
....
public ModifiedART2(float a, float b, float e, float p, float theta,
                    float delta, int inputSize, float tolerance)
{
    _f1Layer = new F1Layer(a, b, e, p, theta, delta, inputSize, tolerance);
    _f2Layer = new F2Layer();
    winnerList = new List<int>();
    winningTimes = new List<int>();
    winningTimes.Add(0);
}
....
```