DOI: 10.2352/ISSN.2470-1173.2016.15.IPAS-188

Refractory Neural Nets and Vision – A Deeper Look

Thomas C. Fall; Kalyx Associates; Los Gatos, CA, USA

Abstract

In an earlier paper, it was shown that the neuron's refractory period (the period of time after the neuron has fired before it can fire again) can serve as a short term local memory. In particular, if an array of refractory neurons (the retina) trains over an image, is then offset, the trained pixel comparisons to the offset pixels are done globally across the entire array. The refractory period is biologically based, and so is the offset; the offset is done by ocular microtremors. Together, they provide a tool that can do grey scale boundary and texture segmentation.

This paper significantly extends the capabilities of refractory neural nets by pointing out that refractory neurons can be arranged into XOR gates. We have a 'pixel-predictor' and use an XOR gate to compare the sensing of a pixel to the prediction for that pixel. If the two are the same, then nothing comes up from the gate. If they are different, then a signal comes out and a modification is made to the pixel-predictor. These predictors can be done at multiple levels of coarseness which effectively give us a multilayer classifier, i.e., a deep learning capability.

Overview and Background

Artificial Neural Networks (ANN) are core computational engines in the machine learning toolbox. The vision community uses ANNs to achieve learning from massive learning sets and have garnered successes. ANNs are based on biological facts, namely that learning in biologic neural systems can achieved by reinforcement of appropriate synaptic connections. Using this approach, ANNs have parsed through massive data sets to find patterns that were not visible to the humans. However, the biological neurons have more story to tell. Namely they have a refractory period, a period of time after the neuron has fired before it can fire again, which is about 1 msec [1]. From the overall network perspective, if the neuron has fired, it is no longer in the network- this means the topology of the network has changed. This can be a much stronger effect than just modifying the synapse strength. But what makes it powerful is that these neurons, with their refractory nature, can combine together to do significant processing of the raw sensory data stream that can enhance the ANN process. In particular, it can locally perform the synapse weighting updates that backpropagation would provide.

The use of refractory characteristics in neural nets could provide a significant new approach. I presented a paper, "Refractory Neural Nets and Vision", [2] at the 2014 Electronic Imaging Conference where it was shown that the neuron's refractory period (the period of time after the neuron has fired before it can fire again) can serve as a short term local memory. In particular, if an array of refractory neurons (the retina) that has trained over an image is now offset, then pixel comparisons of the trained to the offset is done globally across the entire array. Biologically, the offset is done by ocular microtremors (OMTs). [3] In that paper, we showed they provided a tool that can do grey scale boundary and texture segmentation. Figure 1 shows a test image done for the previous paper where we had two regions side by side, on fading from black on top to medium grey, the other fading from that same medium grey to white. Thresholding would put one of those medium grey regions in the wrong segment. Figure 2 shows that the RNN with the Ocular Microtremor can pick out the edge between these regions.

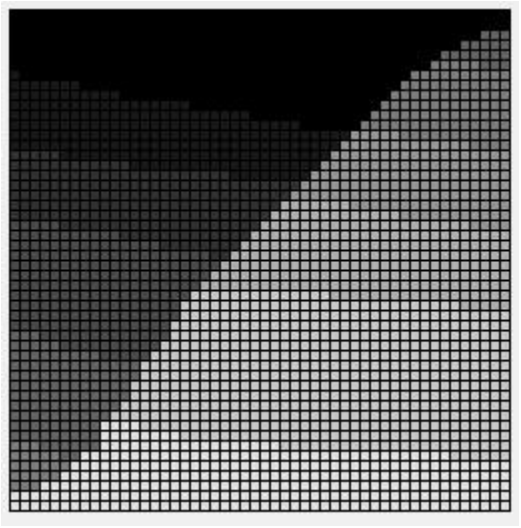

Figure 1: Test RNN/OMT Process

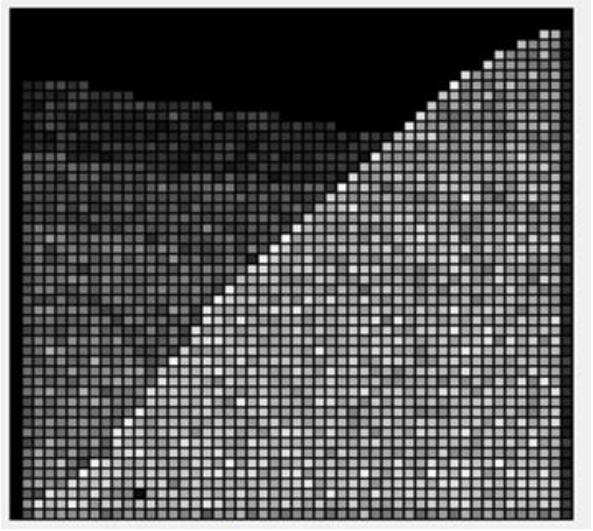

Figure 2: The RNN and OMT discovers the Edge

We see the edge revealed by the 'flash' effect: the brighter region sees even brighter pixels on its side; the darker region darker.

DOI: 10.2352/ISSN.2470-1173.2016.15.IPAS-188

This fact of edge detection facilitated by OMT could well have bearing on one of the mysteries of vision, namely the perception of 'forbidden' colors such as 'yellowish blue' or 'redish green' in defiance of Hering's laws of color opponency. Billock, et.al report on experiments where they "*…used a dual Purkinje image eye tracker to retinally stabilize bipartite color fields whose hues and achromatic border contrast were controlled*." [4] Note that if there is an OMT, the image stabilizer would shift the image to obviate the OMT movement so the shift that does the edge detection does not happen. With, equiluminant opposing colors, they report the segmentation disappears and since there is now no boundary, each color floods into the other's region. Billock et al report that perceptions of this were varied: *Our subjects ... were tongue-tied in their descriptions of these colors, using terms like ''green with a red sheen,'' or ''red with green highlights.''* We will expand on this in the later section 'RNN Pixel Predictor'.

This paper will expand on that by investigating how the RNN approach can learn from images and how it can develop classifiers at several levels in an unsupervised fashion, providing a deep learning capability. Jordan and Mitchell in a recent article in Science [5] that surveys machine learning point out that "The most widely used machine-learning methods are supervised learning methods." Further, they state "One high-impact area of progress in supervised learning in recent years involves deep networks, which are multilayer networks of threshold units, each of which computes some simple parameterized function of its inputs." This paper will discuss a proof of concept study that indicates the RNN approach may provide the deep learning for vision in an unsupervised way

Methodology

The RNN approach is biologically inspired, but our aim in this paper is to investigate the computational impacts, not the biological. The computational architecture is a series of arrays where lower layers feed into higher layers. We take a cellular automata/discrete event simulator (DES) approach where the control mechanism is time clicked (the clicks are nominally 0.5 msec, but we don't have a clock – we loop from one click to the next). So all of the arrays are updated at the same simulation clock time. Cellular automata would compute updates for all elements of all arrays. We bring in DES philosophy and only run the update code for those array elements that have event changed inputs.

The objective was 'proof of concept', not 'image processing production code', so we used Python instead of C++ or Java. Python is fantastic for algorithm development. It is interpreted instead of compiled, so one can immediately see the impact of changes.

In the previous paper, we had two computational layers: the retinal layer and the aggregation layer. At the bottom was the image layer. It was fixed and fed into the retinal layer. The retinal layer is an array of neurons that react to image layer pixel values directly below them and that array can be offset. These neurons are implemented as finite state machines with the pixel value as input. The state transitions for a given retinal neuron, R(m,n) are defined below:

If R(m,n) = 2 R(m,n) → 1 (1)

If R(m,n) = 1 R(m,n) → 0 (2)

If R(m,n) = 0 (3)

If random(N) < Image(i,j)

R(m,n) → 2 and

firing transmitted to Aggregation(i,j)

Else R(m,n) → 0

These state transitions define the following behavior. If the retinal neuron, R(m,n), is in state 2 or state 1, it cannot fire. If it is in state 0, it goes through a stochastic determination based on the brightness of Image(i,j) as to whether it fires or not. If not, it remains fire-able. If it does fire, it goes into the first part of the latency period (state 2) and the fact of firing is carried to the (i,j)th elements of the aggregation layers where we aggregate over several OMTs. To reiterate, upon an OMT, the information in the (i,j)th elements of the image get processed by the retinal layer into the (i,j)th elements of the aggregation layer. The aggregation layer in a sense becomes a model of the image layer as developed by the retinal layer.

The retinal layer is positioned on top of the image and each retinal neuron will react to the image pixel below it. If the image pixel is very bright, if the neuron is fireable, it will likely fire immediately. Then all those fired neurons will go into their refractory state, so all those will be unable to fire. There may be some unfired neurons of the offset array that over the bright area that do fire, but altogether, the firings of neurons that had been over the bright areas would be significantly lower. Conversely, if the retinal neurons had been over a dark area, they would likely not to have fired, so if the OMT moves them to over a bright area, a high proportion of these will likely fire. Thus we should see a significant increase in the firing rate compared to what we see in the more interior bright region – colloquially it would be said we "see a flash." And on the dark side, the OMT effect will produce darker output on that side of the boundary.

At start up, we accelerate the stabilization by randomly setting the refractory, retinal neurons to be in the various refractory states. That way, at startup, only a portion will be fire-able and on the next click, another portion will become fire-able. We still give it a few clicks to completely stabilize before we start the experiments.

For the operation of the RNN, let's return to the biology. Remembering that what we are expecting is a flash when a movement of the retinal array moves a portion of the retinal array from dark to light, we ask is there a biological process that does this. And there is indeed; it is the ocular microtremor (OMT) and it is 30 Hz to 120 Hz peaking at around 83Hz. [3] This would equate to about 12 msec per microtremor. Given that the absolute refractory period is 1 msec [1] we have about 12 refractory cycles per microtremor. At two clicks per refractory period, this would be 24 clicks per microtremor. In the experiments discussed in the earlier paper [2] we had used 16 clicks per retinal array movement and that seemed to give us pretty good stabilization after the flash. If the neuron fires, that is passed up to the aggregation layer indexed with the position of the image layer. There will be several aggregation arrays. Typically, these only get inputs from the retinal layer upon a microtremor movement (a 'flash'). There are arrays that only get inputs for particular microtremor movements. There would be arrays for 1 pixel up or down, 2 pixels up or down, 1 pixel left or right, etc. That earlier paper pointed out that these aggregate layers were important for texture discovery. Passing a firing up to an aggregational layer increments the value at the pixel index position of that layer.

Another important aspect is that there are no cross-connections within the retinal layer. This means that different portions of the processing of the retinal layer could be easily distributed, since it doesn't require any information flow to other elements. Similarly for the image itself. There may be cross connections at some of the aggregational layers, but even these would be fairly local. Lower level aggregations could be pipe-lined into higher level pipes using a stream-based approach. In other words, we don't have long scale correlations which confound distributed processing approaches. All

DOI: 10.2352/ISSN.2470-1173.2016.15.IPAS-188

this says that this approach is very amenable to Big Data techniques such as analytics, Hadoop MapReduce techniques, etc.

Let us here do a quick comparison to standard edge detection, namely Canny edge detection. From a couple of on-line tutorials, the Canny edge detection can be described as follows. [6] [7] A Guassian blur is applied to smooth out noise. A Sobel operator is applied to find the maximal gradients. Those pixels are used as the starting point of building the edge line. In contrast, since the RNN is a stochastic process, the Gaussian blurring is not necessary. In fact, it would wipe out the fine texture areas RNN detects coincidently with edge detection. [8] Since the RNN process enhances the contrast between the boundary pixels on each side, the boundary lines can often be picked out by just doing differences in the horizontal and the vertical directions.

This short term memory effect of the refractory period can also be of consequence in other portions of the computational process. Figure 3 shows a graphic from an earlier paper [2] that shows how to build an XOR circuit from refractory neurons. Here is a quote from that paper as to how it functions:

"To see the operation, a signal comes to neuron A. It gets transmitted to neuron A which fires (the heavy line indicates the link has sufficient weight to fire the neuron by itself) as well as to B_1. From A, the signal gets transmitted to both A_1 and A_2, which will both fire, if A_1 has not already fired. The dashed links leading from them to A_C show that these are half weight links and both must fire, at the same time, for A_C to fire. If a signal comes in to both A and B at the same time, both A_1 and B_1 will fire a beat early, so they will not be able to fire in concert with their partner. Both signals get squelched. However, a signal coming in on just one of them will get passed through." Later in this paper, we will walk through this process.

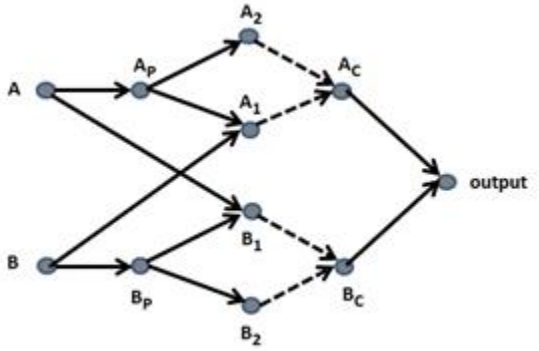

Figure 3: The XOR Circuit Built From RNN Components

These XOR circuits can then be used to implement learning. For each pixel, we have an XOR circuit with the sensed value as one input and the predicted value as another. If they are the same, either both fired or both unfired, there is no output. If they are different there is and that initiates the learning process. If the sensed value is 1 and the pedicted value is 0, then we increment the pixel-predictor for that pixel. For the other case, we would decrement the pixel-predictor.

This XOR circuit is done by wiring together 11 extremely simple processors – that is they are defined by the three simple finite state machine rules above. If the retinal array was NxN, we would have an array of NxN XORs comprised of 11 neurons each. If we physically had these, since they could all run in paralel, the processing would be extremely quick. However, we don't; this has to be implemented as a virtual array where we virtualy traverse the array, one XOR at a time and sequence them onto the CPU. Run time grows as the square of the dimension; our test images are approximately 150x150, small enough for fast computation. These are also fine for proof of concept since the images are so stark it is easy to grasp what the processing is doing.

Refractory Neural Nets – Deep Learning

Refractory Neural Net Adaptable Topologies

That the Refractory Neural Nets can modify their topologies in response to different conditions is a novel computational tool with perhaps wide spreading utility. We will explicate this by doing stepped snapshots of the behavior of an RNN XOR circuit, expanding on our earlier comments. We will have a sequence of four figures showing the side by side (well actually up and down) comparison of the XOR behavior given the two inputs (from Image and from Prediction) 'Agree" (the top, showing both 'A' and 'B' firing at input) and 'Disagree' (the bottom, showing 'A' firing and 'B' not firing.) Let's see how these each play out.

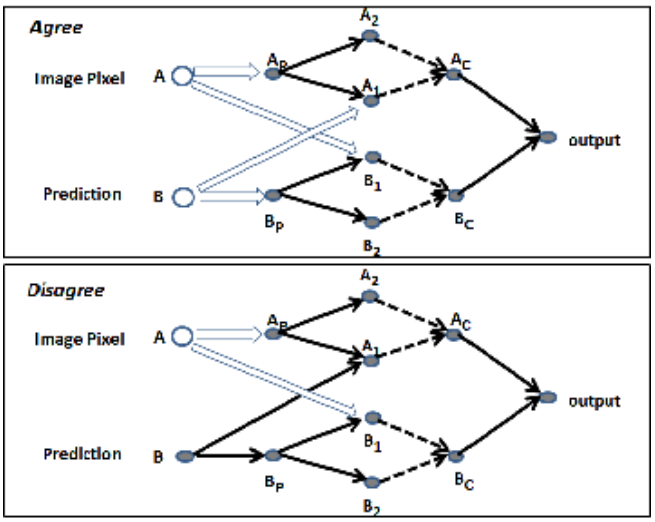

Figure 4: First Step of Exercise of XOR - Agree vs disagree

In the top illustration of Figure 4: First Step of Exercise of XOR - Agree vs disagree, we see both Neuron A and Neuron B firing. Each fires back into its own subnetwork, e.g. from 'A' to 'A_P,', 'B to B_P'. But each also fires into the other network, namely 'A' to 'B_1', 'B' to 'A_1'. The bottom shows only 'A' fires and 'B' does not: that is, there is disagreement between the two inputs.

The top illustration of Figure 5: Next Set Of Neurons Fire, shows the consequences of agreement between the two inputs: both 'A_1' and 'A_P,' fire and both 'B_1' and 'B_P,' fire. Thus from this point on, neither 'A_1' nor 'B_1' can be fired as they are both in their refractory states. The bottom panel shows the consequences of disagreement. 'A' fires and B does not. Since 'B' did not fire, 'A_1' does not get excited and does not fire. Thus 'A_1' is still fireable.

©2016 Society for Imaging Science and Technology
DOI: 10.2352/ISSN.2470-1173.2016.15.IPAS-188

Figure 6: Third Stage Where Agreeing inputs Interfere with Each Other, shows how the two agreeing inputs interfere with each other. 'A_2' fires but 'A_1' does not since it had fired on the previous beat. Similarly, 'B_2' fires but 'B_1' does not. On the lower panel, we see that since B did not fire, 'A_1' did not fire. Thus, when the wave front got there, both 'A_1' and 'A_2' could fire.

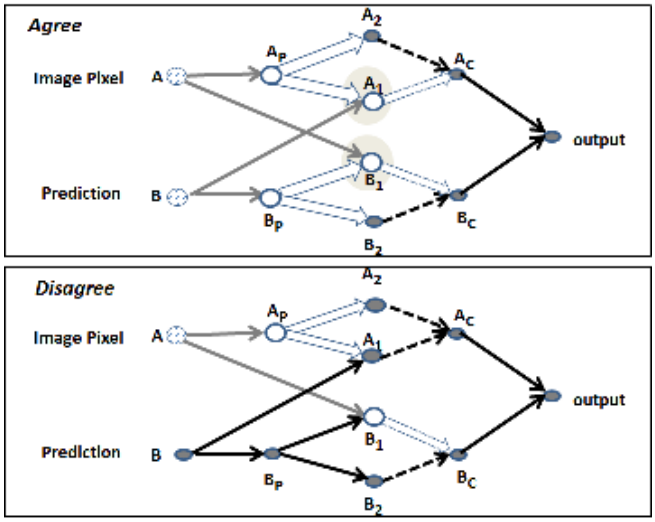

Figure 5: Next Set Of Neurons Fire

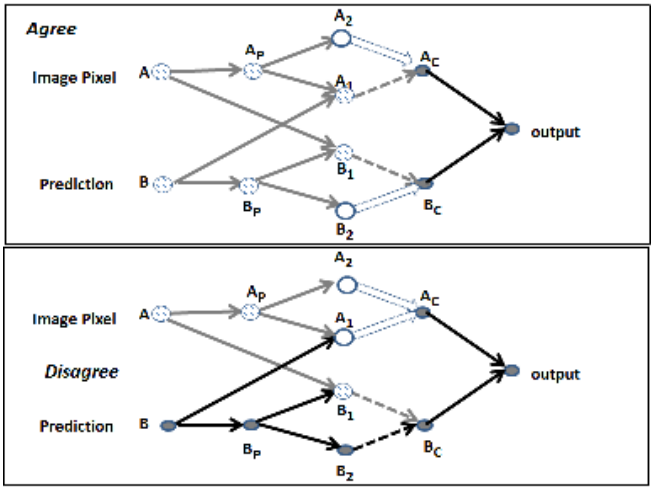

Figure 6: Third Stage Where Agreeing inputs Interfere with Each Other

Figure 7: The Agrees extinguish Each Other top illustration shows since the pulses from 'A_1' and 'A_2' did not arrive simultaneously, 'A_C' did not fire. Similarly for 'B_C'. That is, if the two inputs agree, they will extinguish each other. The lower illustration shows that if only 'A' fires, there will be a consequent pulse at the output. With this, we can detect differences between predicted and the image at pixel levels. The next section will discuss the implications of this in more detail.

Another facet of this that is novel and perhaps a promising line of further examination is that effectively, the agreeing inputs cause an effective, temporary change to the network topology, as is seen in Figure 8: Topology Changes for Agree. Literature searches of Neural Nets have not shown any use of the refractory period in this way, particularly for use in vision processing. S.K.Aityan in a 1994 colloquium abstract [9] mentions the refractory neurons can be assembled into units that do all the binary logic functions including XOR, but I could find nothing that further expanded on this.

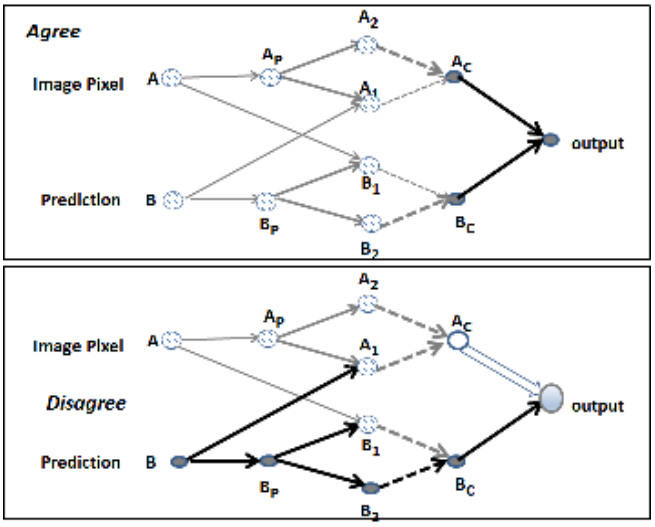

Figure 7: The Agrees extinguish Each Other

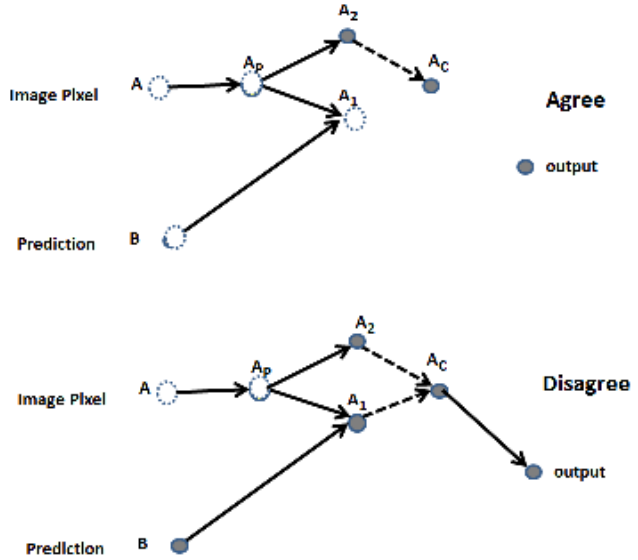

Figure 8: Topology Changes for Agree

The top panel of Figure 8 shows how the topology of the 'A' network is changed by the early firing of 'A_1' due to excitement

©2016 Society for Imaging Science and Technology
DOI: 10.2352/ISSN.2470-1173.2016.15.IPAS-188

from 'B'. This effectively deletes the link from 'A_1' to 'A_C' for the length of time of the refractory period. The bottom panel shows the 'A' network is unchanged if 'B' is not also fired at the same time as 'A'.

From a biological perspective, this XOR could have benefits in that activity only continues further up the line if what is seen is different than what is expected. This decreases the amount of input into the higher levels, giving those the space to handle the idiosyncratic issues of non-normal events. Importantly from a biological perspective is that less energy is required. From a computational algorist view, we might ask how the energy requirements of the pixel-predictor are minimized.

The Refractory Neural Net Pixel-Predictor

The XOR circuit is used to develop and refine pixel-predictors. In the current versions, the pixel-predictors are arrays sized the same as the retinal array and have non-negative integer values. When presented with a new image, we also start with a pixel-predictor array which is zeroed out. We do the clicks of the stabilization period, do the ocular microtremor (OMT) and collect flashes in the appropriate cell of the aggregate layer. The pixel-predictor at this same time makes a prediction as to whether a given pixel would flash by making a stochastic choice based on the integer value in that position. Both of these values are presented to the XOR and if there is no output (that is, both are the same), then there is no change to the pixel- predictor. If not, then the pixel-predictor value is decremented if the pixel-predictor indicated the pixel fired and the pixel didn't. Conversely, if the pixel-predictor didn't fire but the pixel actually did, then the pixel-predictor would be incremented. Since the firing process is stochastic, noise will slow down convergence of the pixel-predictor, but will not kill it. Once we have a pixel-predictor, we can make programmatic changes to the predictor output to compare to other manifestations of the image.

For proof of concept purposes, we will utilize four images that will help delineate this study's problem space, two star shapes (Figure 9: The Star Shapes-4 point and 6 point) and two regular polygons (Figure 10: Regular polygons: Square and Hexagon). These are straightforward non-convex vs convex shapes. These images are fed to the initial process which does the edge detection as we've described above.

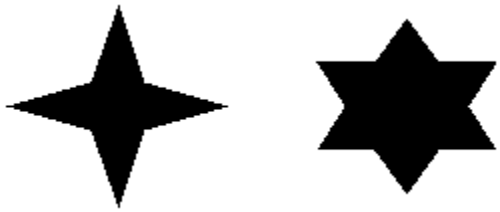

Figure 9: The Star Shapes-4 point and 6 point

Figure 10: the Regular polygons-Square and Hexagon

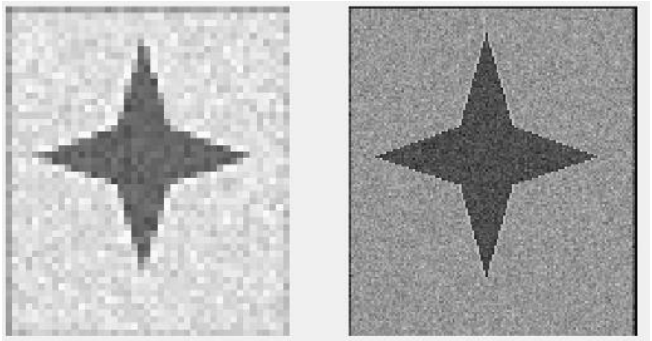

Figure 11: Coarse vs Fine processing

Deep Learning and Classification

In an earlier section, we saw how pixel-predictors would be developed by the using the RNN XOR circuit to do pixel by pixel comparison of the pixel-predictor to the presented image, updating the pixel-predictor as appropriate. This comparison process can tell us of the degree of match between the presented image and any of the predictors currently in inventory. However, in real circumstances, there will be such a plethora of different predictors, that this detail level comparison is not a feasible approach. However, we address this by augmenting the process that builds the pixel-predictors to also build coarser level predictors. Figure 11 illustrates the original fine level pixel-predictor and its coarser version.

These coarser versions are built at the same time as the pixel-predictors. Namely, if we were to increment (decrement) pixel-predictor[i, j], we would increment (decrement) coarse-pixel-predictor[int(i/3), int(j/3)]. This builds a coarser array comprised of 3x3 elements. This reduces the number of compare operation by an order of magnitude.

DOI: 10.2352/ISSN.2470-1173.2016.15.IPAS-188

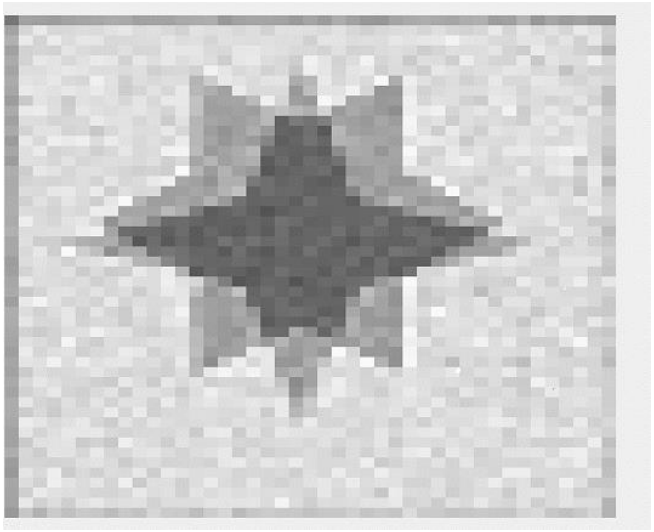

Figure 12: 4-star and 6-star combined

.

Further, we can make coarse predictors that are a blend of several coarse images. Figure 12 shows a blend of the 4-star and the 6-star images. The dark area in the center is where both images are black; the gray areas around that are where one is black and the other is background white. And around that is where both images are white background. Figure 13 shows the blend of square and hexagon.

Given a training set of images, they could be processed for edge detection at the finest level, producing the different coarser level views as a side effect. Figure 11 shows the development of two levels of processing for the 4-star. The edge detection clearly shows on the fine level product (right hand side) where we see the pixels on the boundary on the bright side are much brighter than the other brights and the pixels on the dark side are darker than rest. In the image on the left of Figure 11, the coarser view (responses were binned into 3x3 pixel buckets) we can see this boundary effect, but just barely.

This smoothing allows us to compare in a defocused way. Things will kind of look like others if we can ignore details

So for a sample image, S, the classifier could first look at the coarsest version of S, S_0, and compare that to the coarse predictors for best match. Each of those are the result of a merging of a set of predictors at the next level of fineness that 'look like each other'. Once we have found the best matching coarse level predictor, we can then look at its finer level predictors and find which best match to our sample image in its next finer resolution, S_1

This hierarchical approach, coarser to finer, provides an exponential lever on operational complexity. Effectively, this multilayered hierarchy of increasingly finer pixel-predictors has the same effectiveness as 'deep networks' Jordan and Mitchell [5] allude to

[Author's aside: Infants have a very low visual acuity that gets better – could it be that they are developing these coarser level predictors and don't even try the fine level (even though they could physically parse the image at that finer level]

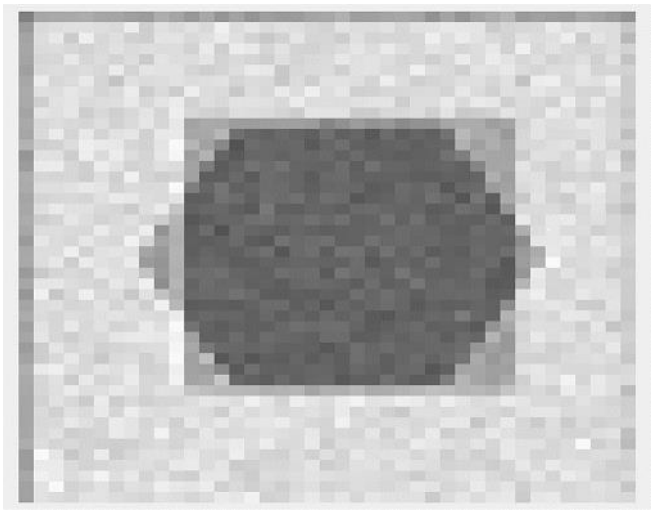

Figure 13: Hex and Square combined

Let's look at how this hierarchy of predictors might be achieved. Figures 12 and 13 show the aggregated predictors for the 4-star/6-star aggregate and the hex/square aggregate. Figure 13, for example, is the result of taking the predictor for the coarse hex and the predictor for the coarse square and merging them as images. That is, each pixel is the average of that pixel position for the hex and for the square. I.e. HS[i,j] = (H[i,j] + S[i,j])/2. If we added on a third image, A, then HSA[i,j] = ((HS[i,j]*2) + A[i,j])/3, each image contributing a 1/3 share. The goal is to find a partition of the training images at their coarsest level so that the partitions differ little internally, but differ significantly with other partitions. This of course is classic clustering. The results of merging within each cluster a coarse image that can act as the exemplar for that cluster.

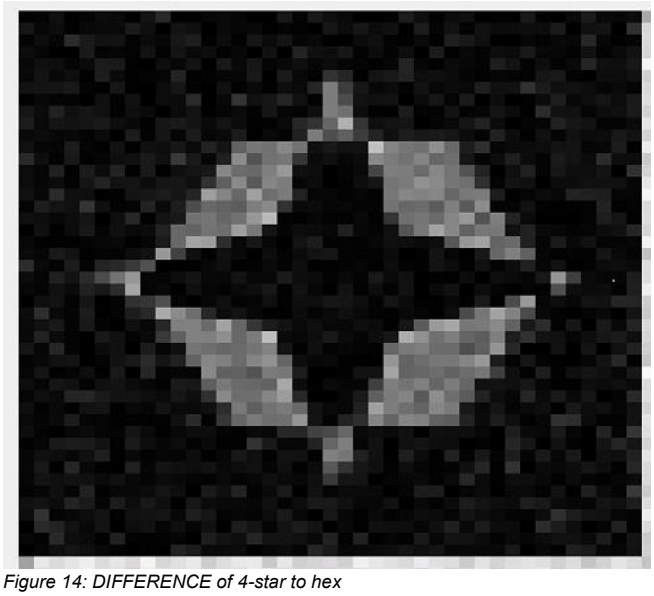

Figure 14: DIFFERENCE of 4-star to hex

DOI: 10.2352/ISSN.2470-1173.2016.15.IPAS-188

Figure 15: Difference of 4-star and 6-star

It is desired to do this clustering in an unsupervised fashion. The training images (coarse level versions) are presented to merge process in sequence. The incoming image is compared to the current exemplars using the difference function. Figures 14 and 15 above show this difference function. The exemplar with the smallest difference is chosen as the cluster and the incoming image is merged into that. However, if the differences to all the current exemplars is too large, the incoming image becomes the initial exemplar of a new cluster. Once we believe we have converged on the exemplar set at the coarsest level, we can take the images in a cluster and build up the exemplar set for that cluster at the finer level. The Merge process provides us a hierarchical classifier which is 1) done unsupervised and 2) robust to noise (since the base processes are stochastic)

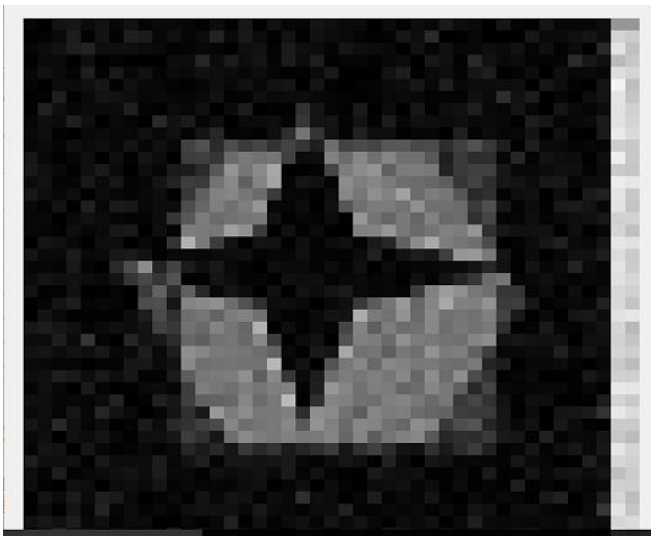

Figure 16: 4-star on the hex-square exemplar

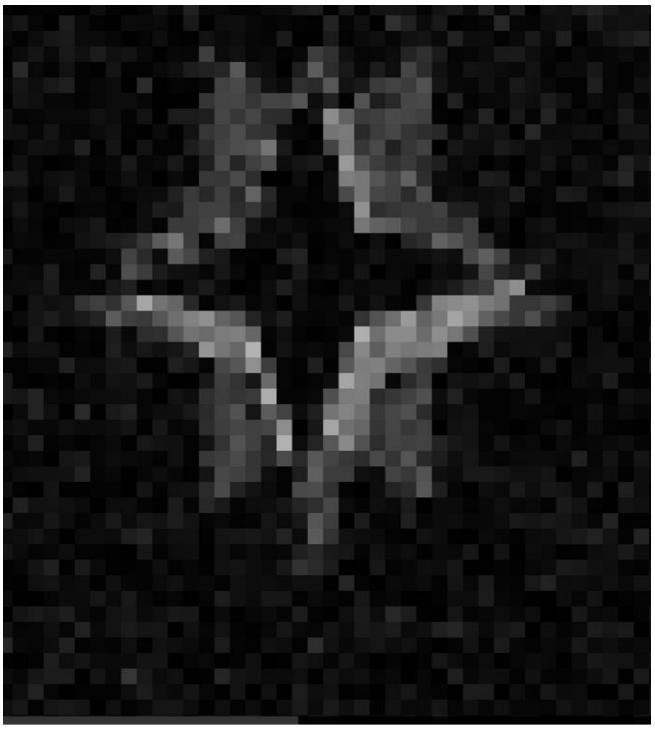

Figure 17: Small 4-star on the 4-star-6-star exemplar

Discussion and Further Direction

Buesing et al in their recent paper [10] start off with the following statement:

"Attempts to understand the organization of computations in the brain from the perspective of traditional, mostly deterministic, models of computation, such as attractor neural networks or Turing machines, have run into problems: Experimental data suggests that

DOI: 10.2352/ISSN.2470-1173.2016.15.IPAS-188

neurons, synapses, and neural systems are inherently stochastic, especially in vivo, and therefore seem less suitable for implementing deterministic computations."

The authors of that article look to defining a probabilstic based analysis of a neural computational network They include the refractory period as part of the Markov Chain Monte Carlo process. However, they do not in that paper consider that the refractory period can actually change the topology of the network. Even though this change only lasts a millisecond, for the events in that msec, that modified topology is what they see and the outputs of that network reflect the input values and the modified topology.

Thus we have a possible new approach to computation: 'dynamic network intelligence' where the network's topology is affected by the inputs and the history, as well as the synapse weightings of the ANN-like components. This approach is inherently stochastic, so fits into reality based processes.

Enhancements envisoned for the RNN code set include the separation of segment boundary pixel-prediction from segment region color/texture pixel-prediction and the development of rotational and dilational programatics. Achievng the first, the boundary detection, would definitely help with the second since we could then deal with substantially smaller sets.

Summary

This paper has discussed several novel insights that could be of use in the image processing toolbox. The refractory period can provide short term memory which, combined with the ocular microtremor gives us a local convolution-like process globally over the retinal array. Texture determination is a by-product of this edge detection. The refractory period can cause temporary changes to the vision network topology; in particular XOR gates. The XOR gates compare what is predicted to what is seen. Differences can be used to modify the predictor. This can be done at several nested layers of coarseness, providing a deep learning capability.

References

[1] G. Ritchison, " http://people.eku.edu/ritchisong/301notes2.htm," [Online].

[2] T. Fall, "Refractory Neural Nets and Vision," in *Image Processing:Algorithms and Systems XII*, San Francisco, CA, 2014.

[3] M. Al-Kabani, E. Mihaylova, N. Collins, V. Toal, D. Coakley and G. Boyle, "Ocular microtremor laser speckle metrology," in *Proc. SPIE 7176, Dynamics and Fluctuations in Biomedical Photonics VI*, San Jose, CA, 2009.

[4] G. a. T. Billock, "Perception of forbidden colors in retinally stabilized equiluminant images: an indication of softwired cortical color opponency?," *Journal of the Optical Society of America A,* Vols. Vol. 18, , no. Issue 10, pp. pp. 2398-2403 , 2001.

[5] T. M. Jordan and M. I. Mitchell, "Machine learning: Trend, perspectives, and prospects," *Science,* vol. 349, no. 6245, pp. 255-260, 2015.

[6] B. Green, "Canny Edge Detection Tutorial," 2002. [Online]. Available: http://dasl.mem.drexel.edu/alumni/bGreen/www.pages.drexel.edu/_weg22/can_tut.html. [Accessed January 2016].

[7] P. Fuller, "Gabor Filter – Image processing for scientists and engineers, Part 6," 23 December 2012. [Online]. Available: http://patrick-fuller.com/gabor-filter-image-processing-for-scientists-and-engineers-part-6/. [Accessed January 2016].

[8] T. C. Fall, "Stochastic neural nets and vision," in *SPIE Vol. 1468 Applications of Artificial Intelligence*, Orlando, FL, 1991.

[9] S. K. Aityan, "Introduction to Refractory Neural Netorks," 5 October 1994. [Online]. Available: ccm.ucdenver.edu/colloq/9495/aityan. [Accessed January 2016].

[10] L. Buesing, J. Bill, B. Nessler and W. Maass, "Neural Dynamics as Sampling: A Model for Stochastic Computation in Recurrent Networks of Spiking Neurons," *PLoS Computational Biology,* vol. 7, no. 11, pp. 1-22, 2011.

Author Biography

Tom Fall received his BS in Physical Chemistry from UC Berkeley and then his Ph.D. in Mathematics also from UC Berkeley. He worked network architecture issues at Lockheed Martin and currently is a consultant with Kalyx Associates for network analysis