Training of Recurrent Internal Symmetry Networks by Backpropagation

Alan Blair and Guanzhong Li

Abstract—**Internal Symmetry Networks are a recently developed class of Cellular Neural Network inspired by the phenomenon of internal symmetry in quantum physics. Their hidden unit activations are acted on non-trivially by the dihedral group of symmetries of the square. Here, we extend Internal Symmetry Networks to include recurrent connections, and train them by backpropagation to perform two simple image processing tasks.**

Keywords – cellular neural networks; group representations; internal symmetry; recurrent dynamics

I. Introduction

Cellular Neural Networks (CNN) are a class of architecture consisting of a large number of identical neural networks arranged in a cellular array [1]. Recent years have seen a growing interent in CNNs, particularly for image processing tasks [2]. Generally, the weights of a CNN are determined by hand-crafted design, or by global random search. Comparatively little research has been done on the training of CNNs by backpropagation [3]. Here, we report on the training, by backpropagation, of a particular class of CNN known as *Internal Symmetry Networks* (ISN), which employ a special kind of weight sharing scheme inspired from quantum physics. Feedforward ISNs have previously been trained by TD-learning to perform move evaluation for the game of Go [4]. In the present work, we extend the ISN framework to include recurrent connections, and test it on two simple image processing tasks – edge detection (using feedforward ISNs) and a simplified image segmentation task (using recurrent ISNs).

I. Internal Symmetry Networks

Consider a Cellular Neural Network comprised of a large number of identical recurrent neural networks arranged in a cellular array (see Fig 1). For image processing tasks, each pixel in the image will generally correspond to one cell in the array. For clarity of exposition, we assume a square image of size *n*-by-*n*, with $n = 2k+1$. The array can then be considered as a lattice Λ of vertices $\lambda=[a,b]$ with $-k \le a,b \le k$. It will be convenient to denote by $\underline{\Lambda}$ the "extended" lattice which includes an extra row of vertices around the edge of the image, i.e. $\underline{\Lambda} = \{[a,b]\}_{-(k+1) \le a,b \le (k+1)}$.

We can define a local neighborhood around any given vertex $\lambda=[a,b]$ by specifying a set of increments or "offset" values which determine the position, *relative to* λ, of all vertices in the neighborhood. Here, we will employ two such neighborhood structures, denoted by $\mathcal{M}$ and $\mathcal{N}$:

$$\mathcal{M} = \{[0,0], [1,0], [0,1], [-1,0], [0,-1]\},$$

$$\mathcal{N} = \mathcal{M} \cup \{[1,1], [-1,1], [-1,-1], [1,-1]\}$$

When viewed as offsets from a particular vertex, $\mathcal{M}$ represents the vertex itself plus the neighboring vertices to its East, North, West and South; $\mathcal{N}$ includes these but also adds the diagonal vertices to the North-East, North-West, South-West and South-East (see Fig 1).

Each cell $\lambda = [a,b] \in \Lambda$ will have its own set of input, hidden and output units denoted by $I^{[a,b]}$, $H^{[a,b]}$ and $O^{[a,b]}$. Each off-edge cell $\lambda = [a,b] \in \underline{\Lambda}\backslash\Lambda$ also has input and hidden units, but no output. The entire collection of input, hidden and output units $\mathcal{I}$, $\mathcal{H}$ and $\mathcal{O}$ for the whole network can thus be written as:

$$\mathcal{I} = \{I^{[a,b]}\}_{[a,b]\in\underline{\Lambda}}$$

$$\mathcal{H} = \{H^{[a,b]}\}_{[a,b]\in\underline{\Lambda}}$$

$$\mathcal{O} = \{O^{[a,b]}\}_{[a,b]\in\Lambda}$$

Each cell $\lambda \in \Lambda$ is connected to its nine $\mathcal{N}$-neighboring cells (i.e. those in a local 3×3 neighborhood) with input-to-hidden connections V_{HI} , hidden-to-output connections V_{OH} and input-to-output connections V_{OI} ; in addition, each cell is connected to its five $\mathcal{M}$-neighboring cells by recurrent hidden-to-hidden connections V_{HH} . We will use B_H and B_O to represent the "bias" at the hidden and output units. The standard recurrent neural network update equations then take this form:

$$H_{new}{}^{\lambda} \leftarrow H(\mathcal{I},\mathcal{H}_{old})^{\lambda} = \tanh(B_H + \textstyle\sum_{\nu\in\mathcal{N}} V_{HI}{}^{\nu} I^{\lambda+\nu} + \sum_{\mu\in\mathcal{M}} V_{HH}{}^{\mu} H_{old}{}^{\lambda+\mu})$$
$$O^{\lambda} \leftarrow O(\mathcal{I},\mathcal{H}_{new})^{\lambda} = \phi\ (B_O + \textstyle\sum_{\nu\in\mathcal{N}} V_{OI}{}^{\nu} I^{\lambda+\nu} + \sum_{\nu\in\mathcal{N}} V_{OH}{}^{\nu} H_{new}{}^{\lambda+\nu})$$

where ϕ is the sigmoid function $\phi(z)=1/(1+e^{-z})$.

Alan Blair and Guanzhong Li are with the School of Computer Science and Engineering, University of New South Wales, Sydney 2052, Australia.
(email: blair@cse.unsw.edu.au)

We assume that for the off-edge cells ($\lambda \in \underline{\Lambda}\backslash\Lambda$) the hidden units H^λ remain identically zero, while the inputs I^λ take on special values to indicate that they are off the edge of the image. We assume a discrete-time model, with the hidden unit activations of all cells being updated synchronously.

Many image processing tasks are invariant to geometric transformations of the image (rotations and reflections) as well as being shift-invariant (with appropriate allowance for "edge effects"). With this in mind, we design our system in such a way that the network updates are invariant to these transformations. A number of different weight sharing schemes have previously been proposed [5]. In the present work, we employ a recently developed weight sharing scheme known as Internal Symmetry Networks [4], based on group representation theory.

The group $\mathcal{G}$ of symmetries of a (square) image is the dihedral group D_4 of order 8. This group is generated by two elements r and s – where r represents a (counter-clockwise) rotation of 90° and s represents a reflection in the vertical axis (see Fig 2). The action of D_4 on Λ (or $\underline{\Lambda}$) is given by

$$r\,[a, b] = [-b, a]$$

$$s\,[a, b] = [-a, b] \qquad (1)$$

Assuming the action of $\mathcal{G}$ on $\mathcal{N}$ (or $\mathcal{M}$) is also defined by these equations, it is clear that for $g \in \mathcal{G}$, $\lambda \in \Lambda$ and $\nu \in \mathcal{N}$,

$$g(\lambda + \nu) = g(\lambda) + g(\nu).$$

Any element $g \in \mathcal{G}$ acts on the inputs $\mathcal{I}$ and output units $\mathcal{O}$ by simply permuting the cells:

$$g(\mathcal{I}) = \{\, I^{g[a,b]} \,\}_{[a,b]\in\Lambda}$$

$$g(\mathcal{O}) = \{\, O^{g[a,b]} \,\}_{[a,b]\in\Lambda}$$

In addition to permuting the cells, it is possible for $\mathcal{G}$ to act on some or all of the hidden unit activations within each cell, in a manner analogous to the phenomenon of *internal symmetry* in quantum physics. The group D_4 has five irreducible representations, which we will label as *Trivial*(T), *Symmetrical*(S), *Diagonal*(D), *Chiral*(C) and *Faithful*(F). They are depicted visually in Fig 3, and presented algebraically via these equations:

$$r(T) = T, \quad s(T) = T$$

$$r(S) = -S, \quad s(S) = S$$

$$r(D) = -D, \quad s(D) = -D$$

$$r(C) = C, \quad s(C) = -C$$

$$r(F)_1 = -F_2, \quad s(F)_1 = -F_1$$

$$r(F)_2 = F_1, \quad s(F)_2 = F_2$$

We consider, then, five types of hidden units, each with its own group action determined by the above equations. In general, an

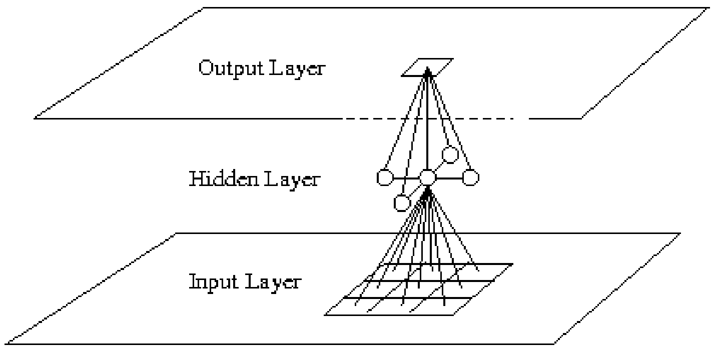

Figure 1. System Architecture

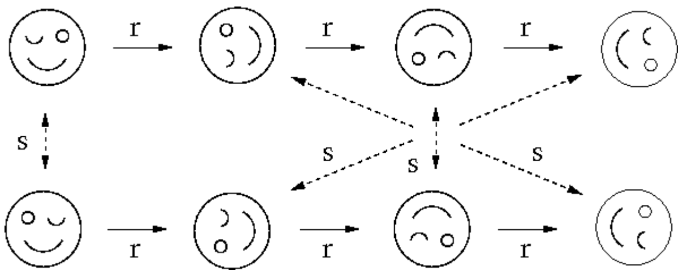

Figure 2. The dihedral group D_4 with generators r, s

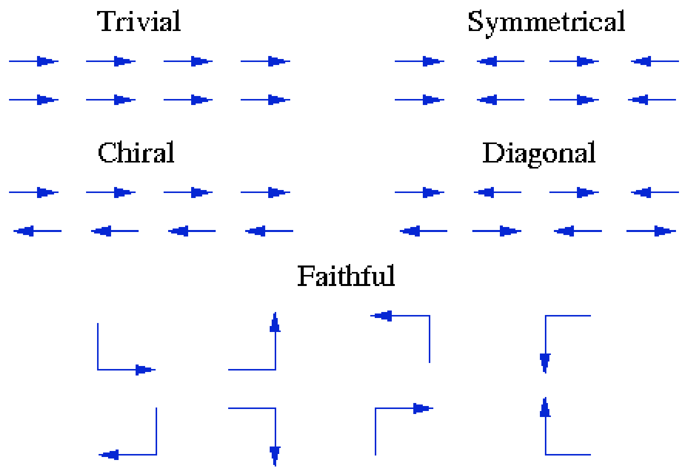

Figure 3. The five irreducible representations of D_4

ISN can be characterized by a 5-tuple specifying the number of each type of hidden node at each cell (i_T,i_S,i_D,i_C,i_F). Because it is 2-dimensional, hidden units corresponding to the Faithful representation will occur in pairs (F_1, F_2) with the group action "mixing" the activations of F_1 and F_2. The composite hidden unit activation for a single cell then becomes a cross-product

$$H = T^{i_T} \times S^{i_S} \times D^{i_D} \times C^{i_C} \times (F_1 \times F_2)^{i_F}$$

with the action of $\mathcal{G}$ on $\mathcal{H}$ given by

$$g(\mathcal{H}) = \{g(H^{g[a,b]})\}_{[a,b]\in\underline{\Lambda}}$$

We want the network to be invariant to the action of $\mathcal{G}$ in the sense that for all $g \in \mathcal{G}$,

$$g(\mathcal{H}_{\text{new}}(\mathcal{I}, \mathcal{H}_{\text{old}}) = \mathcal{H}_{\text{new}}(g(\mathcal{I}), g(\mathcal{H}_{\text{old}}))$$

$$g(\mathcal{O}(\mathcal{I}, \mathcal{H}_{\text{new}}) = \mathcal{O}(g(\mathcal{I}), g(\mathcal{H}_{\text{new}}))$$

This invariance imposes certain constraints on the weights of the network, which are outlined in the Appendix.

I. Experiments

We test the ISN framework on two simple image processing tasks. For black and white images, the network has two inputs per pixel. One input encodes the intensity of the pixel as a grey scale value between 0 and 1. The other input is a dedicated "off-edge" input which is equal to 0 for inputs inside the actual image, and equal to 1 for inputs off the edge of the image (i.e. for vertices in $\underline{\Lambda}\backslash\Lambda$). This kind of encoding could in principle be extended to color images by using four inputs per pixel (three to encode the R,G,B or Y,U,V values, plus the dedicated "off-edge" input).

A. Edge Detection using Feedforward ISNs

For our first experiment, we test whether an ISN can be trained to perform Canny Edge Detection [6]. In this case, there is only one output unit for each pixel, and the aim is for the network to reproduce, as well as possible, the result of a Canny edge detection algorithm applied to the original image. This experiment is intended as a "proof of concept", and as a vehicle for tuning the parameters of our system and exploring different hidden node configurations, before introducing the additional complexity of recurrent connections.

A number of combinations of parameters and hidden units were tried. The best results were obtained using cross entropy minimization, with a learning rate of 5×10^{-9}, momentum of 0.9 and hidden unit configuration of (3,1,1,0,1). Cross entropy means that, if O_λ is the output and T_λ is the target, then the cost function to be minimized is

$$-\textstyle\sum_\lambda T_\lambda \log O_\lambda + (1-T_\lambda) \log(1-O_\lambda)$$

Fig 4 shows the training and test set error (per pixel) for 600,000 training epochs. Fig 5 shows the input, target and network output for one of the test images.

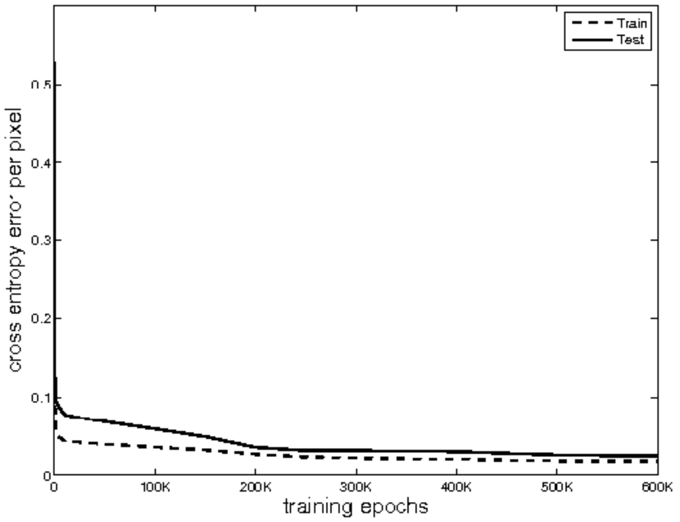

Figure 4. Cross entropy training error (dotted) and test error (solid) per pixel, for Canny edge detection task.

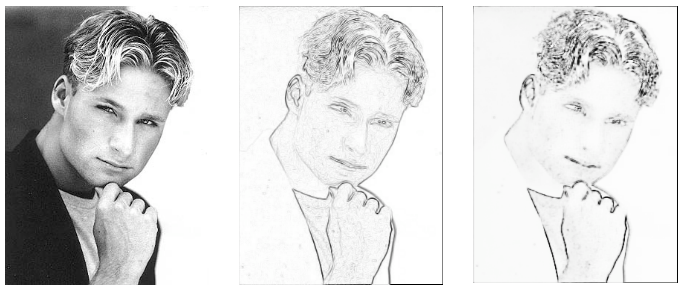

Figure 5. Example of original test image (left), target image (center) and network output (right).

A. Wallpaper Segmentation using Recurrent ISNs

It order to explore the recurrent dynamical aspects of the ISN framework, we devised a "wallpaper segmentation" task. This is a simplified version of image segmentation, where each image is a patchwork of different styles of "wallpaper", each consisting of an array of small motifs on an otherwise blank canvass.

The training images and test image for this task are shown in the top row of Fig 6. The network has 4 outputs - one for each style of wallpaper. During training, the target value at each pixel is 1 for the k^{th} output and 0 for the other outputs, if the pixel belongs to a part of the image corresponding to the k^{th} style of wallpaper. During testing, the largest of the 4 outputs for each pixel is taken to be the network's prediction of which style of wallpaper is present in that part of the image. The test image combines all four styles, and the spacing between the motifs is slightly larger for the test image than for the training images.

For each input image, the ISN is applied for 10 cycles. At each cycle, the hidden unit activations for all cells are updated synchronously, with the new values depending on the inputs and (recurrently) on the values of neighboring hidden nodes at the previous time step.

A purely feedforward CNN would not be able to perform this task, because the output at each pixel would depend only on the portion of the image in a small neighborhood of that pixel, so the blank areas of the image would fail to be correctly classified.

There have been many studies into the training of recurrent neural networks by backpropagation for predicting or recognizing symbol sequences [7]. Such training often leads to

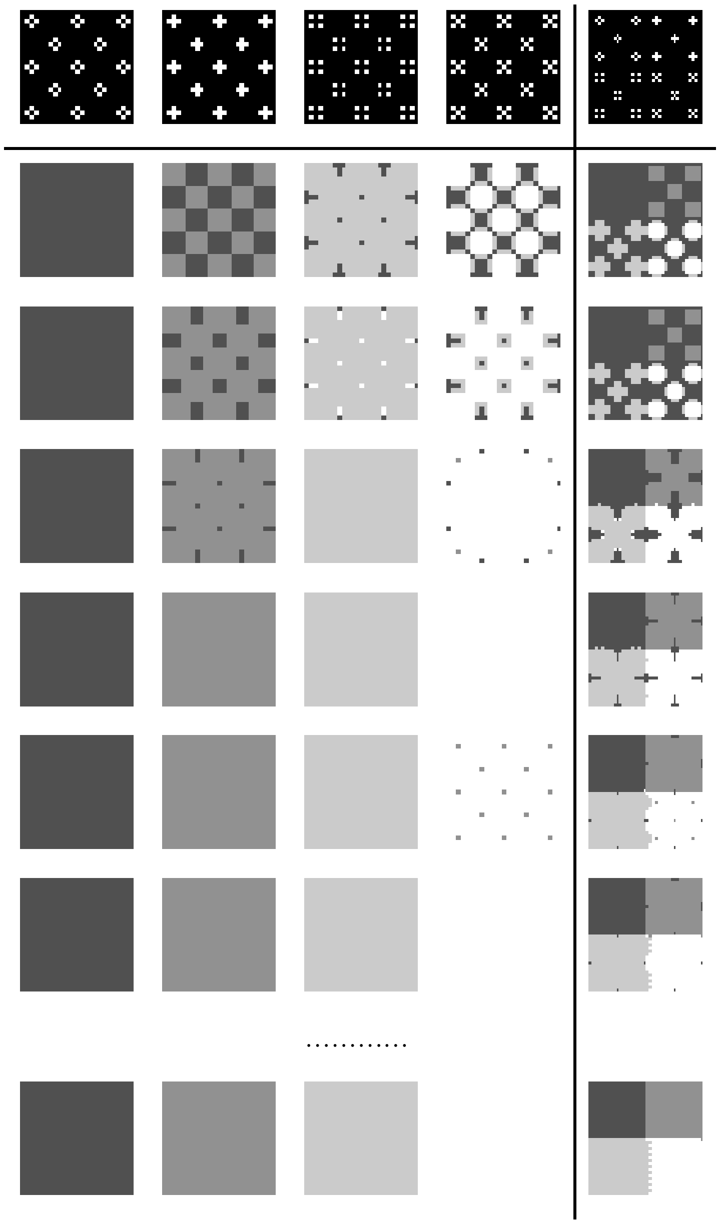

Figure 6. Training images and test image for wallpaper segmentation (top row) and output of the network at epoch 890K for cycles 1 to 6 (middle rows) and cycle 10 (bottom row).

instabilities, where the task is continually learned but then unlearned [8]. It has generally been found that the training is more stable when the backpropagation is applied only for a single time step and not "unrolled" [9,10]. This effect is even more noticeable in the case of ISNs, where the activations spread in two dimensions. In our early experiments, with backpropagation unrolled to several previous cycles, we observed a resonance effect occurring after a few thousand epochs, in which the backpropagated differentials suddenly blow up to very large values. To avoid this problem, we always limit the backpropagation to a single cycle.

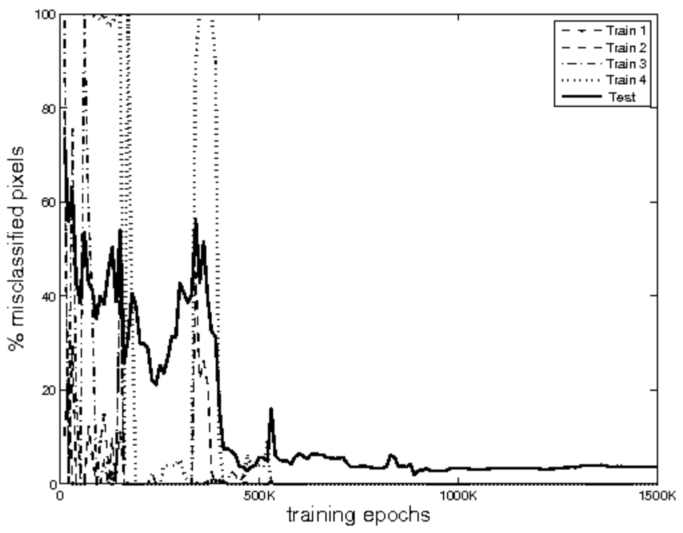

Figure 7. Percentage of misclassified pixels on training images and test image, for wallpaper segmentation task.

We found that the best results were obtained using cross entropy minimization, with a learning rate of 5×10^{-8}, momentum of 0.3 and hidden unit configuration of (4,0,0,0,0). Fig 7 shows the percentage of misclassified pixels during 1.5 million epochs of training. Misclassification on the training images undergoes several initial fluctuations, but finally reaches zero after 539K epochs and remains zero thereafter. The number of misclassified pixels on the test image continues to fall, reaching a minimum of 28 (from a total of 1444 pixels) between 890K and 896K, but then increases to around 50.

Fig 6 shows the classification provided by the network at epoch 890K, for cycles 1 to 6 and cycle 10. For the training images, the network classifies all pixels correctly at cycle 4, shows slight misclassification at cycle 5, but returns to correct classification for cycles 6 to 10. For the test image, the number of misclassified pixels continues to drop, reaching a minimum of 28 pixels by Cycle 10.

Fig 8 plots the activations of three of the four hidden units, at epoch 890K, for the training images (left) and the test image (right) at cycles 1, 2, 5 and 10. Pixels belonging to Class 1, 2, 3, 4 are assigned the symbols $\times$, O, Δ ,∇ (respectively).

The hidden unit activations for pixels of the test image (right) follow similar trajectories to those of the training images, but with some divergence. When we examine the output at Cycles 5 and 10 in Fig 6 we see that, at Cycle 5, some Class 4 pixels have been misclassified as Class 2 (for both the training and test image) and that, at Cycle 10, several Class 4 pixels in the test image have been misclassified as Class 2 or 3. The hidden unit activations for these pixels are visible as ∇'s in Fig 8. In both cases, there appear to be overlaps between the other symbols as well, which means that the network must be using the remaining hidden unit, or the hidden units of neighboring cells, in order to correctly classify these pixels.

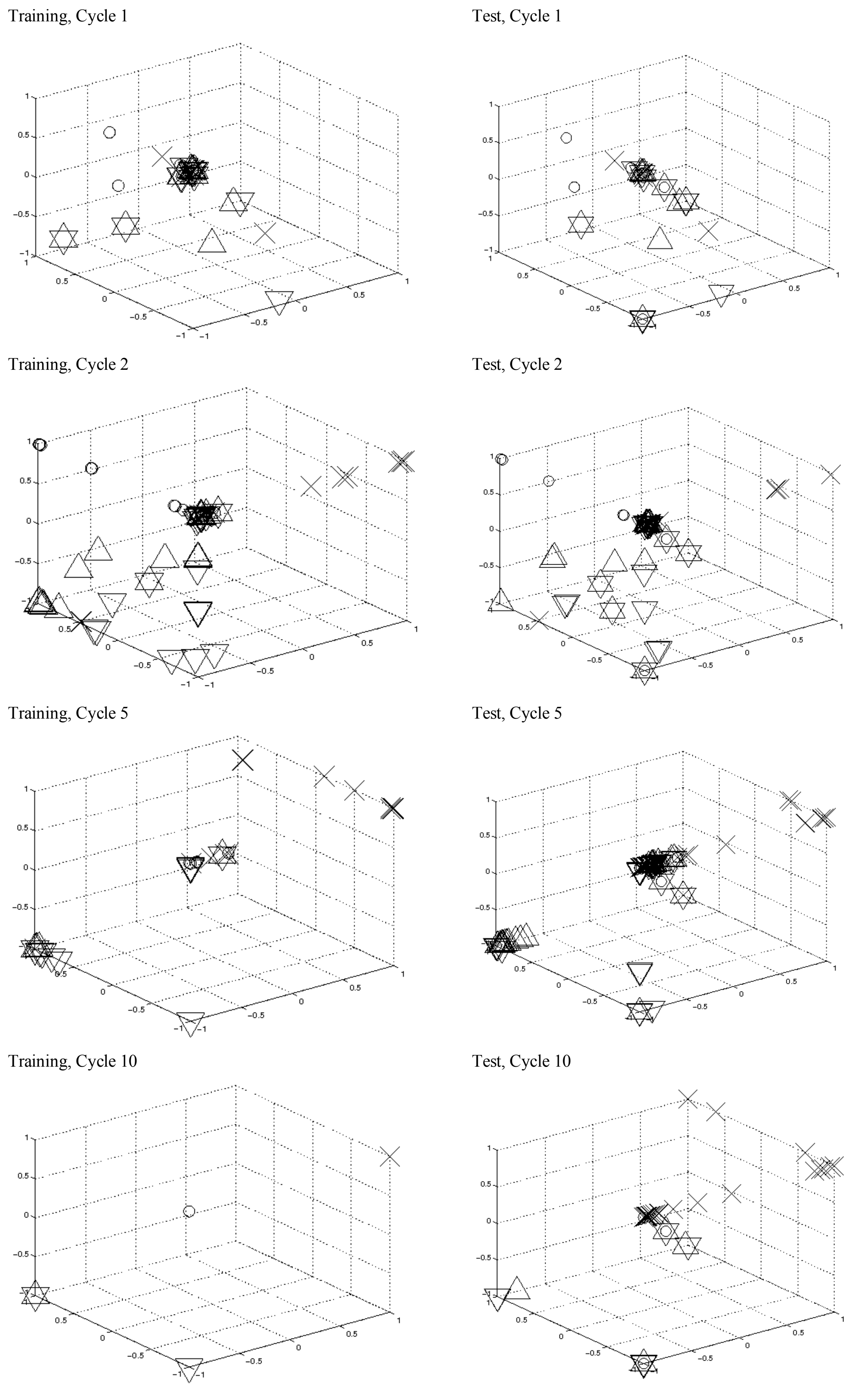

Figure 8. Hidden unit activations of network at epoch 890K, for cycles 1, 2, 5 and 10.

II. Conclusion

We have shown that Internal Symmetry Networks can be successfully trained by backpropagation to perform two simple image processing tasks. When recurrent connections are included, stability becomes an issue; however, successful training can be achieved, provided the learning rate is sufficiently low (and the number of training epochs correspondingly large).

For the edge detection task, a combination of different types of hidden units was found to give the best result. However, for the wallpaper segmentation task, a configuration including only the *Trivial* type of hidden unit appeared to be more effective. One possible reason for this is that the hidden units were only connected to neighboring input cells within a small (3×3) neighborhood. This small neigborhood, combined with the symmetry constraints, meant that the *Symmetrical* and *Diagonal* hidden units were connected to only 4 inputs each, compared to 9 inputs for the *Trivial* hidden units. In ongoing work, we are extending our approach to include connections to a larger (5×5) neighborhood, in which case the *Symmetrical* and *Diagonal* units would be connected to 16 inputs (compared to 25 inputs for the *Trivial* units). We plan to test whether this larger neighborhood would shift the balance in the relative potency of the various hidden unit types.

Appendix – Weight Sharing

A. Feedforward Connections

$$V^{\nu}_{OH} = [\ V^{\nu}_{OT}\ V^{\nu}_{OS}\ V^{\nu}_{OD}\ V^{\nu}_{OC}\ V^{\nu}_{OF_1}\ V^{\nu}_{OF_2}\ ]$$
$$V^{\nu}_{HI} = [\ V^{\nu}_{TI}\ V^{\nu}_{SI}\ V^{\nu}_{DI}\ V^{\nu}_{CI}\ V^{\nu}_{F_1I}\ V^{\nu}_{F_2I}\ ]^T$$
$$V^{E}_{OI} = V^{N}_{OI} = V^{W}_{OI} = V^{S}_{OI}\ ,\ V^{NE}_{OI} = V^{NW}_{OI} = V^{SW}_{OI} = V^{SE}_{OI}$$
$$V^{E}_{OT} = V^{N}_{OT} = V^{W}_{OT} = V^{S}_{OT}\ ,\ V^{NE}_{OT} = V^{NW}_{OT} = V^{SW}_{OT} = V^{SE}_{OT}$$
$$V^{E}_{TI} = V^{N}_{TI} = V^{W}_{TI} = V^{S}_{TI}\ ,\ V^{NE}_{TI} = V^{NW}_{TI} = V^{SW}_{TI} = V^{SE}_{TI}$$
$$V^{O}_{OF_2} = V^{O}_{OF_2} = V^{O}_{OF_1} = V^{O}_{OF_2} = 0$$
$$V^{E}_{OF_1} = V^{N}_{OF_2} = -V^{W}_{OF_1} = -V^{S}_{OF_2} = V^{E}_{F_1I} = V^{N}_{F_2I} = -V^{W}_{F_1I} = -V^{S}_{F_2I}$$
$$V^{E}_{OF_2} = V^{N}_{OF_1} = V^{W}_{OF_2} = V^{S}_{OF_1} = V^{E}_{F_2I} = V^{N}_{F_1I} = V^{W}_{F_2I} = V^{S}_{F_1I} = 0$$

$$V^{NE}_{OF_1} = -V^{NW}_{OF_1} = -V^{SW}_{OF_1} = V^{SE}_{OF_1},\ V^{NE}_{OF_2} = V^{NW}_{OF_2} = -V^{SW}_{OF_2} = -V^{SE}_{OF_2}$$
$$V^{NE}_{F_1I} = -V^{NW}_{F_1I} = -V^{SW}_{F_1I} = V^{SE}_{F_1I}\ ,\ V^{NE}_{F_2I} = V^{NW}_{F_2I} = -V^{SW}_{F_2I} = -V^{SE}_{F_2I}$$
$$V^{E}_{OS} = -V^{N}_{OS} = V^{W}_{OS} = -V^{S}_{OS}\ ,\ V^{NE}_{OD} = -V^{NW}_{OD} = V^{SW}_{OD} = -V^{SE}_{OD}$$
$$V^{E}_{SI} = -V^{N}_{SI} = V^{W}_{SI} = -V^{S}_{SI}\ ,\ V^{NE}_{DI} = -V^{NW}_{DI} = V^{SW}_{DI} = -V^{SE}_{DI}$$
$$V^{\mu}_{OD} = V^{\mu}_{DI} = 0, \qquad \mu \in \{O, E, N, W, S\}$$
$$V^{\nu}_{OS} = V^{\nu}_{SI} = 0, \qquad \nu \in \{O, NE, NW, SW, SE\}$$
$$V^{\nu}_{OC} = V^{\nu}_{CI} = 0, \qquad \nu \in \{O, E, N, W, S, NE, NW, SW, SE\}$$

B. Recurrent Connections

$$V^{\mu}_{HH} =$$
$$\left| \begin{array}{cccccc}
V^{\mu}_{TT} & V^{\mu}_{TS} & V^{\mu}_{TD} & V^{\mu}_{TC} & V^{\mu}_{TF_1} & V^{\mu}_{TF_2} \\
V^{\mu}_{ST} & V^{\mu}_{SS} & V^{\mu}_{SD} & V^{\mu}_{SC} & V^{\mu}_{SF_1} & V^{\mu}_{SF_2} \\
V^{\mu}_{DT} & V^{\mu}_{DS} & V^{\mu}_{DD} & V^{\mu}_{DC} & V^{\mu}_{DF_1} & V^{\mu}_{CF_2} \\
V^{\mu}_{CT} & V^{\mu}_{CS} & V^{\mu}_{CD} & V^{\mu}_{CC} & V^{\mu}_{CF_1} & V^{\mu}_{DF_2} \\
V^{\mu}_{F_1T} & V^{\mu}_{F_1S} & V^{\mu}_{F_1D} & V^{\mu}_{F_1C} & V^{\mu}_{F_1F_1} & V^{\mu}_{F_1F_2} \\
V^{\mu}_{F_2T} & V^{\mu}_{F_2S} & V^{\mu}_{F_2D} & V^{\mu}_{F_2C} & V^{\mu}_{F_2F_1} & V^{\mu}_{F_2F_2}
\end{array} \right|$$

$$V^{E}_{TT} = V^{N}_{TT} = V^{W}_{TT} = V^{S}_{TT}\ ,\ V^{NE}_{SS} = V^{NW}_{SS} = V^{SW}_{SS} = V^{SE}_{SS}$$
$$V^{E}_{TS} = -V^{N}_{TS} = V^{W}_{TS} = -V^{S}_{TS}\ ,\ V^{E}_{DC} = -V^{N}_{DC} = V^{W}_{DC} = -V^{S}_{DC}$$
$$V^{NE}_{ST} = -V^{NW}_{ST} = V^{SW}_{ST} = -V^{SE}_{ST}\ ,\ V^{NE}_{CD} = -V^{NW}_{CD} = V^{SW}_{CD} = -V^{SE}_{CD}$$
$$V^{O}_{TS} = V^{O}_{TF_i} = -V^{O}_{SF_i} = V^{O}_{DC} = V^{O}_{DF_i} = V^{O}_{CF_i} = 0$$
$$V^{O}_{ST} = V^{O}_{F_iT} = -V^{O}_{F_iS} = V^{O}_{CD} = V^{O}_{F_iD} = V^{O}_{F_iC} = 0$$
$$V^{\mu}_{DD} = V^{\mu}_{CC} = 0, \qquad \mu \in \{E, N, W, S\}$$
$$V^{\mu}_{TD} = V^{\mu}_{TC} = V^{\mu}_{SD} = V^{\mu}_{SC} = 0, \qquad \mu \in \{O, E, N, W, S\}$$
$$V^{\mu}_{DT} = V^{\mu}_{CT} = V^{\mu}_{DS} = V^{\mu}_{CS} = 0, \qquad \mu \in \{O, E, N, W, S\}$$
$$V^{E}_{SF_1} = -V^{N}_{SF_2} = -V^{W}_{SF_1} = V^{S}_{SF_2}\ ,\ V^{E}_{TF_1} = V^{N}_{TF_2} = -V^{W}_{TF_1} = -V^{S}_{TF_2}$$
$$V^{E}_{F_1S} = -V^{N}_{F_2S} = -V^{W}_{F_1S} = V^{S}_{F_2S}\ ,\ V^{E}_{F_1T} = V^{N}_{F_2T} = -V^{W}_{F_1T} = -V^{S}_{F_2T}$$
$$V^{E}_{SF_2} = V^{N}_{SF_1} = V^{W}_{SF_2} = V^{S}_{SF_1}\ ,\ V^{E}_{TF_2} = V^{N}_{TF_1} = V^{W}_{TF_2} = V^{S}_{TF_1} = 0$$
$$V^{E}_{F_2S} = V^{N}_{F_1S} = V^{W}_{F_2S} = V^{S}_{F_1S}\ ,\ V^{E}_{F_2T} = V^{N}_{F_1T} = V^{W}_{F_2T} = V^{S}_{F_1T} = 0$$
$$V^{E}_{DF_2} = V^{N}_{DF_1} = -V^{W}_{DF_2} = -V^{S}_{DF_1}\ ,\ V^{E}_{CF_2} = -V^{N}_{CF_1} = -V^{W}_{CF_2} = V^{S}_{CF_1}$$
$$V^{E}_{F_2D} = V^{N}_{F_1D} = -V^{W}_{F_2D} = -V^{S}_{F_1D}\ ,\ V^{E}_{F_2C} = -V^{N}_{F_1C} = -V^{W}_{F_2C} = V^{S}_{F_1C}$$
$$V^{E}_{DF_1} = V^{N}_{DF_2} = V^{W}_{DF_1} = V^{S}_{DF_2}\ ,\ V^{E}_{CF_1} = V^{N}_{CF_2} = V^{W}_{CF_1} = V^{S}_{CF_2} = 0$$
$$V^{E}_{F_1D} = V^{N}_{F_2D} = V^{W}_{F_1D} = V^{S}_{F_2D}\ ,\ V^{E}_{F_1C} = V^{N}_{F_2C} = V^{W}_{F_1C} = V^{S}_{F_2C} = 0$$
$$V^{O}_{F_1F_1} = V^{O}_{F_2F_2} = 0$$
$$V^{E}_{F_1F_1} = V^{N}_{F_2F_2} = V^{W}_{F_1F_1} = V^{S}_{F_2F_2}\ ,\ V^{E}_{F_2F_2} = V^{N}_{F_1F_1} = V^{W}_{F_2F_2} = V^{S}_{F_1F_1} = 0$$
$$V^{\mu}_{F_1F_2} = V^{\mu}_{F_2F_1} = 0, \qquad \mu \in \{O, E, N, W, S\}$$

References

[1] L. Chua and L. Yang, "Cellular Neural Networks: Theory", IEEE Trans. on Circuits and Systems, 35(10), pp 1257-1272, 1988.

[2] L. Chua and T. Roska, "Cellular Neural Networks and Visual Computing", Cambridge University Press, 2002.

[3] W.J. Ho and C.F. Osborne, "Texture Segmentation using Multi-layered Backpropagation", Proc. 1991 Int'l Joint Conference on Neural Networks, pp. 981-986.

[4] A. Blair, "Learning Position Evaluation for Go with Internal Symmetry Networks", Proc. 2008 IEEE Symposium on Computational Intelligence and Games, pp. 199-204.

[5] Y. LeCun et al, "Backpropagation Applied to Handwritten Zip Code Recognition", Neural Computation 1(4), pp. 541-551, 1989.

[6] J. Canny, "A Computational Approach to Edge Detection", IEEE Trans. Pattern Analysis and Machine Intelligence 8, pp 679-714, 1986.

[7] J. Elman, "Finding Structure in Time", Cognitive Science 14(2), pp. 179-211, 1990.

[8] P. Rodriguez, J. Wiles and J. Elman, "A recurrent neural network that learns to count", Connection Science 11(1), pp. 5-40, 1999.

[9] J. Pollack, "The Induction of Dynamical Recognizers", Machine Learning 7, pp. 227-252, 1991.

[10] Y.Bengio, P.Simard and P.Frasconi, "Learning long-term dependencies with gradient descent is difficult", IEEE Trans. Neural Networks 5(2): 157-166, 1994.