Recurrent Internal Symmetry Networks by Backpropagation in Wallpaper Image Segmentation

Guanzhong Li
School of Computer Science & Engineering
University of New South Wales
Sydney, Australia
glix955@cse.unsw.edu.au

***Abstract*—Internal Symmetry Networks are a new developed class of Cellular Neural Network inspired by the phenomenon of internal symmetry in quantum physics. The structures of the nets are based on irreducible group representations. Recurrent cycles are extended to the nets very recently. Dynamic recurrent cycles are proved to be effective by many experiments. In this paper, train them by backpropagation with noise to perform wallpaper image processing tasks.**

Keywords – cellular neural networks; group representations; internal symmetry; dynamic cycle of recurrent;noise

I. Introduction

Cellular Neural Networks (CNN) [1], invented from Cellular automata, are widely used for image processing tasks [2]. Generally, most previous work design weights of a CNN by hand or by global random search. Comparatively little research has been done on the training of CNNs by backpropagation [3]. Here, we report the on the training, by backpropagation, of a particular class of CNN known *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 Then ISN framework is extended to recurrent connections for wallpaper segmentation and performs well. [5] In this paper, *Dynamic Cycle of Recurrent* (DCR) connection strategy is applied based on the same wallpaper image segmentation task. There are many forms of Recurrent Neural Networks (RNN) [6] but DCR is a totally new form. In practice, structural noise with small value added to the training images can increase the accuracy during a limited period. Many studies also show the benefit of adding appropriate noise to training sets. [7] So a class of structural noise is added to the training images. It is found the error rate of the test image has been reduced to 1/6 of the previous work.

II. Internal Symmetry Networks

Consider a Cellular Neural Network comprised of a large number of identical Simple Recurrent (Elman) Networks arranged in a cellular array. 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 λ=[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. $\underline{\Lambda} = \{[a,b]\}_{-(k+1) \le a,b \le (k+1)}$.

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 [4]. In the present work, we employ a recently developed weight sharing scheme known as Internal Symmetry Networks [8], based on group representation theory.

The group G of symmetries of an (square) image is the dihedral group D_8 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 in Appendix C). The action of D_8 on Λ (or $\underline{\Lambda}$) is given by

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

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

We will use M and N to denote neighborhood structures in the form of offset values:

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

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

When viewed as offsets from a particular vertex, M represents the vertex itself plus the neighboring vertices to its East, North, West and South; N includes these but also adds the diagonal vertices to the North-East, North-West, South-West and South-East. Assuming the action of G on N (or M) is also given by (1), it is clear that for $g \in$ G, $\lambda \in \Lambda$ and $\nu \in$ N,

$$g(\lambda + \nu) = \mathrm{g}(\lambda) + \mathrm{g}(\nu).$$

Each cell $\lambda = [a,b] \in \Lambda$ has its own set of input, hidden and output units denoted by $\mathrm{I}^{[a,b]}$, $\mathrm{H}^{[a,b]}$ and $\mathrm{O}^{[a,b]}$. 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 I, H and O for the whole network can thus be written as:

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

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

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

For an individual cell $\lambda \in \Lambda$, the neural network update equations are given by

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

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

In other words, each cell is connected to its neighboring cells by input-to-hidden connections $\mathrm{V_{HI}}$, hidden-to-output connections $\mathrm{V_{OH}}$, input-to-output connections $\mathrm{V_{OI}}$ and hidden-to-hidden (recurrent) connections $\mathrm{V_{HH}}$. $\mathrm{B_H}$ and $\mathrm{B_O}$ represent the “bias” at the hidden and output units. 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. For the experiments reported here, the hidden unit activations of all cells are updated synchronously.

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

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

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

In addition to permuting the cells, it is possible for G to act in 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_8 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 2, and presented algebraically via these equations:

$$r\,(\mathrm{T}) = \mathrm{T},\ s(\mathrm{T}) = \mathrm{T}$$

$$r\,(\mathrm{S}) = -\mathrm{S},\ s(\mathrm{S}) = \mathrm{S}$$

$$r\,(\mathrm{D}) = -\mathrm{D},\ s(\mathrm{D}) = -\mathrm{D}$$

$$r\,(\mathrm{C}) = \mathrm{C},\ s(\mathrm{C}) = -\mathrm{C}$$

$$r\,(\mathrm{F})_1 = -\mathrm{F}_2,\ s(\mathrm{F})_1 = -\mathrm{F}_1$$

$$r\,(\mathrm{F})_2 = \mathrm{F}_1,\ s(\mathrm{F})_2 = \mathrm{F}_2$$

We consider, then, five types of hidden units, each with its own group action determined by the above equations. In general, an ISN can be characterized by a 5-tuple specifying the number of each type of hidden node at each cell ($i_\mathrm{T},i_\mathrm{S},i_\mathrm{D},i_\mathrm{C},i_\mathrm{F}$). 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

$$\mathrm{H} = \mathrm{T}^{i_\mathrm{T}} \times \mathrm{S}^{i_\mathrm{S}} \times \mathrm{D}^{i_\mathrm{D}} \times \mathrm{C}^{i_\mathrm{C}} \times (\mathrm{F}_1 \times \mathrm{F}_2)^{i_\mathrm{F}}$$

with the action of G on H given by

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

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

$$g(\mathsf{H}_{\mathrm{new}}\ (\mathsf{I},\ \mathsf{H}_{\mathrm{old}}) = \mathsf{H}_{\mathrm{new}}\ (g(\mathsf{I}),\ g(\mathsf{H}_{\mathrm{old}}))$$

$$g(\mathsf{O}\ (\mathsf{I},\ \mathsf{H}_{\mathrm{new}}) = \mathsf{O}\ (g(\mathsf{I}),\ g(\mathsf{H}_{\mathrm{new}}))$$

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

III. Dynamic Cycle of RNN

Inspired from the phenomenon of multi-attractors of RNN, about 100 related experiments were tried. It is proved by experiment that adding cycles can reduce the lowest test error, in many cases that meet the requirement of Point 4. Elman Network [9] is arranged in a cellular array. Use a strategy of DCR below:

1.Start to run the RNN via a small number of cycles.

2.Evaluate the performance of task each cycle of each epoch and stop when appropriately.

3.Select an epoch to read the generated weight for initialization and restart to run the RNN, but with a larger number of cycles.

4.Go to Step 2

In step2, the evaluation criteria can be similar to the trajectory of epochs of non-recurrent NN. If under-fitting, the cycle is too small; and if over-fitting, it is too big.

IV. Experiments

ISN framework is test on wallpaper segmentation task. 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 principal 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).

Wallpaper segmentation 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. By experiments on wallpaper, some useful ideas can be extended to texture segmentation.

The training images and test image for this task are shown in the top row of Fig 5. 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.

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). 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 1 shows the classification provided by the network at epoch 890K, for cycles 1 to 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.

Then by evaluation of the trajectory of all recurrent cylces of epoch 890K, it is possible for the test error still to decrease when more cycles are given. So DCRNN is applied here, with 20 cycles.

Figure 2 shows the classification provided by the new network at epoch 2010K, for cycles 1 to 6. (Test error increase after cycle 7 and there are not important improvements after cycle 10). For the training images, the network classifies all pixels correctly after cycle 3. For the test image, the number of misclassified pixels continues to drop, reaching a minimum of 23 pixels by Cycle 6. Fig 3 shows the two output images with the lowest errors from Fig 1 and 2 respectively.

As shown in Fig 4, structural noise is added to the training set. 6 cycles are applied initially. In epoch 4380K, the lowest of number of misclassified pixels on the test image is 20 at cycle 6. Then it increases to around 40 and 50 at cycle 5 and 6 respectively. With another experiment with 10 cycles initially, the lowest number of misclassified pixels on the test image is still at cycle 5 or 6. So DCRNN is applied at epoch 4380K with cycle 10. At cycle 4650K, the number of misclassified pixels on the test image decreases to 5 at cycle 6, and then oscillate around 10.

V. 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 sometimes be achieved, provided the learning rate is sufficiently low (and the number of training epochs correspondingly large).

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 our 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.

A new method DCR for BP is applied for a wallpaper-segmentation task. The accuracy has been improved. The interesting point is the noticeable improvement accuracy emerges in an earlier cycle not the one that has the best accuracy during all the cycles at the restarting point.

In many research of NN, parameter tuning is focused on learning rate, momentum, weight-decay, delta decay and cross entropy, especially the first two. In practice, recurrent cycle is also required to tune. Meanwhile, it is more time consuming for cycle tuning than all the other parameters. DCR provides a strategy to avoid restart from the beginning but some epoch with high evaluation score. Some related experiments show that just adding cycle cannot guarantee to be more efficient. If cycles are set to 20 initially, the missing classified pixels are around 90 in epoch 1.76M. So overfitting produces time consuming but guarantee accuracy, but it is better to use DCR not set too many initial cycles.

By adding some structural noise to the training set, it is excited to see the nearly perfect output.

This experiment uses ISN, a specific CNN with BP. However, overfitting phenomenon in cycles of RNN can be extended to all kinds of BP.

The future work is to analyze these correct restarting conditions and generate some algorithm with more detail. DCR cannot be only combined with 3*3 neighboring, but also 5*5 or larger n*n. Then it is possibly to do some more complicated tasks.

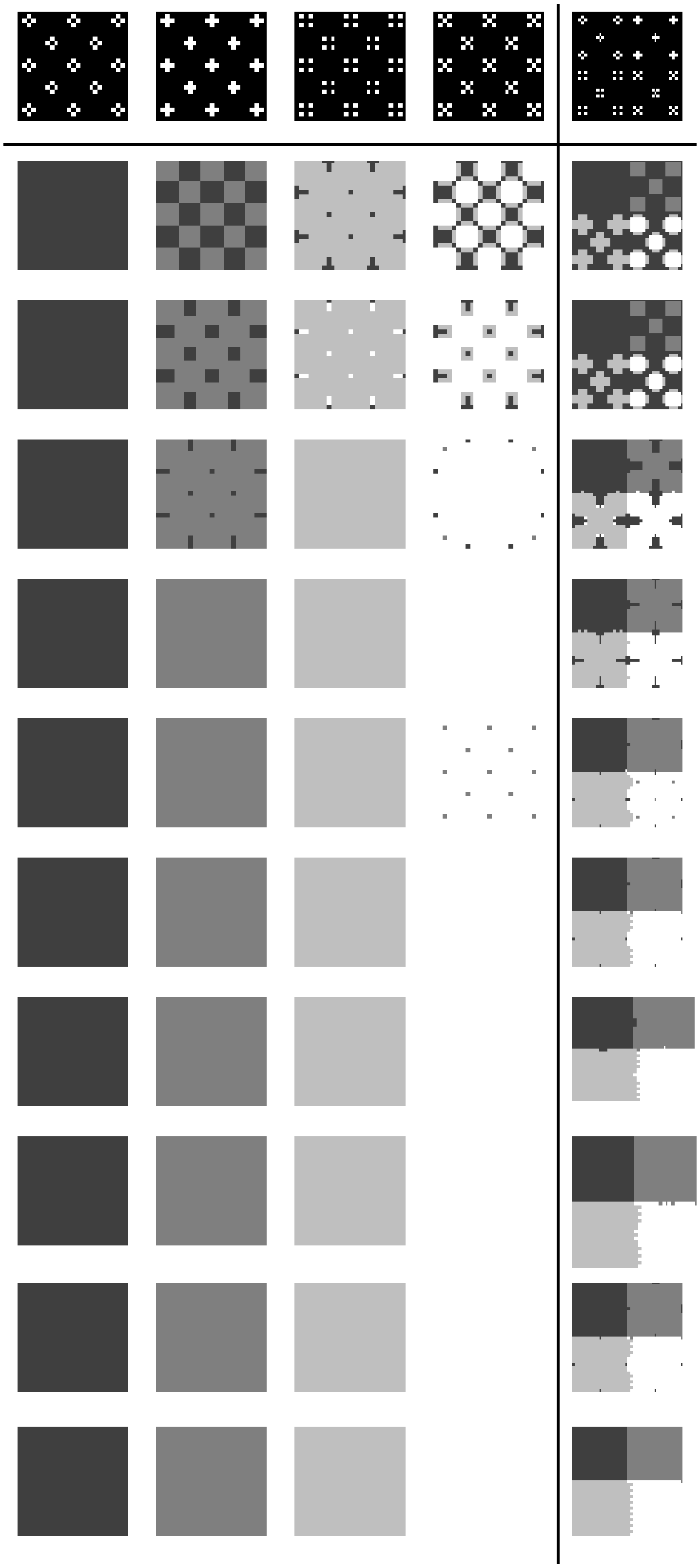


Figure 1

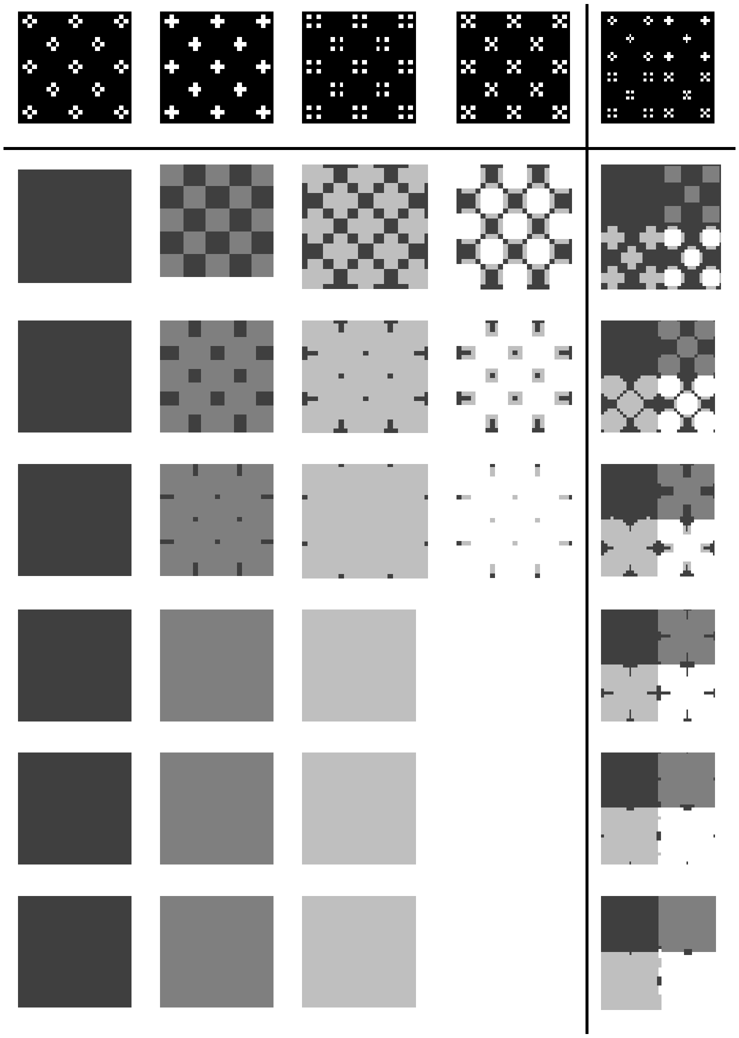


Figure 2

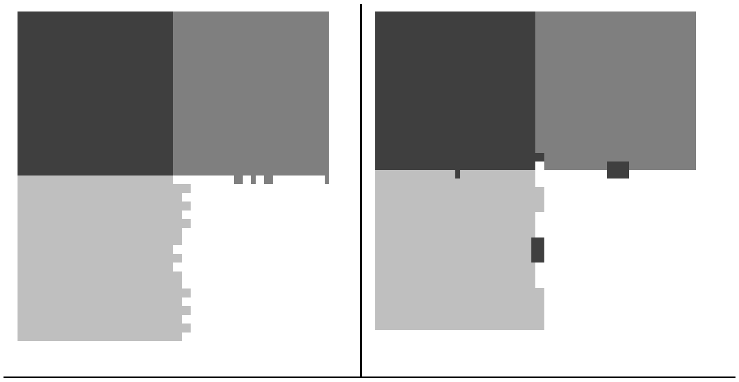


Figure 3

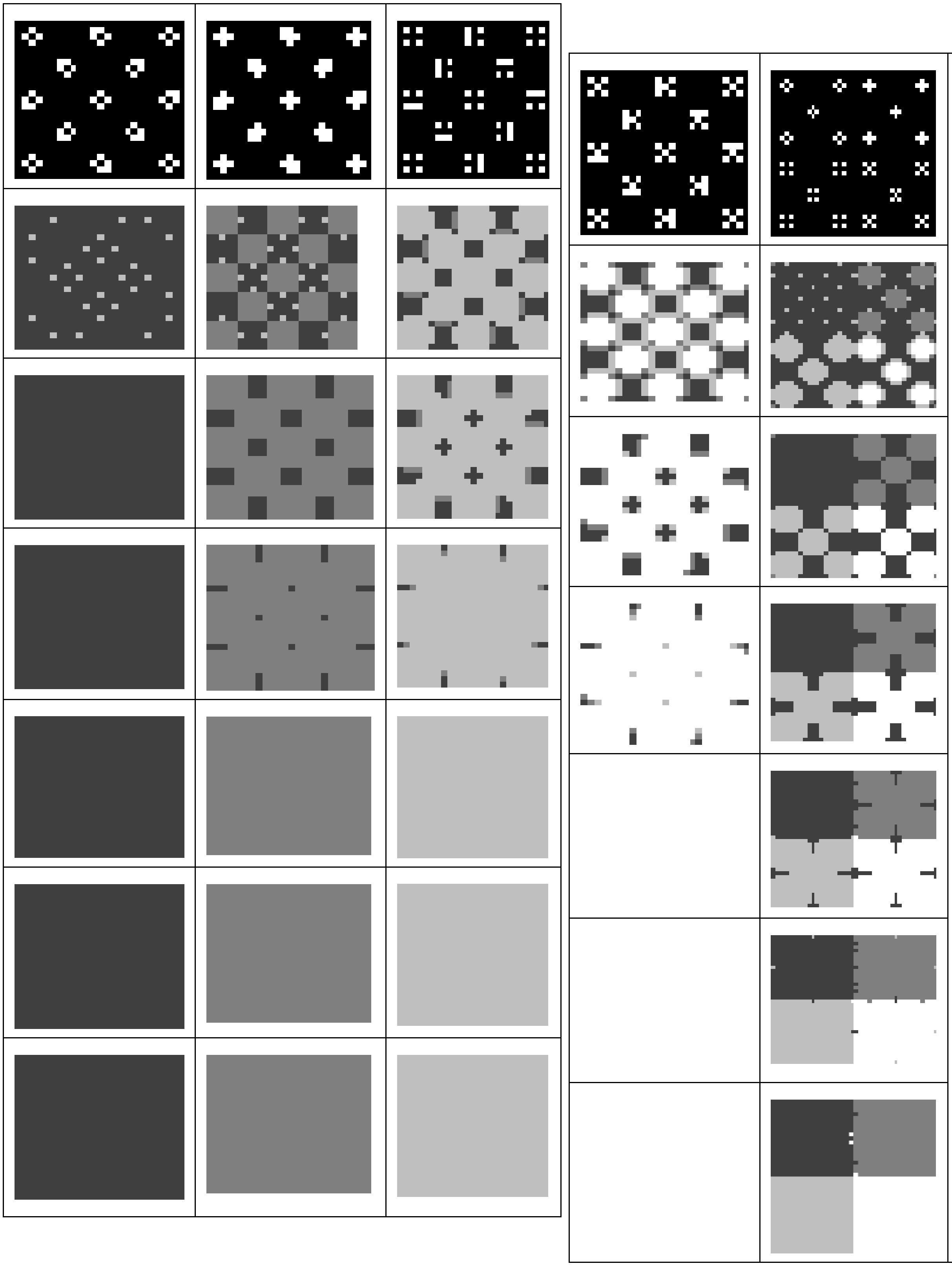


FIG 4

APPENDIX – WEIGHT SHARING

Feedforward Connections

$V^{\nu}_{OH} = [\ V^{\nu}_{OT}\ V^{\nu}_{OS}\ V^{\nu}_{OD}\ V^{\nu}_{OC}\ V^{\nu}_{OF_1}\ V^{\nu}_{OF_2}\ ]$

$V^{\nu}_{HI} = [\ V^{\nu}_{TI}\ V^{\nu}_{SI}\ V^{\nu}_{DI}\ V^{\nu}_{CI}\ V^{\nu}_{F_1I}\ V^{\nu}_{F_2I}\ ]^T$

$V^{E}_{OI} = V^{N}_{OI} = V^{W}_{OI} = V^{S}_{OI}\ ,\ V^{NE}_{OI} = V^{NW}_{OI} = V^{SW}_{OI} = V^{SE}_{OI}$

$V^{E}_{OT} = V^{N}_{OT} = V^{W}_{OT} = V^{S}_{OT}\ ,\ V^{NE}_{OT} = V^{NW}_{OT} = V^{SW}_{OT} = V^{SE}_{OT}$

$V^{E}_{TI} = V^{N}_{TI} = V^{W}_{TI} = V^{S}_{TI}\ ,\ V^{NE}_{TI} = V^{NW}_{TI} = V^{SW}_{TI} = V^{SE}_{TI}$

$V^{O}_{OF_2} = V^{O}_{OF_2} = V^{O}_{OF_1} = V^{O}_{OF_2} = 0$

$V^{E}_{OF_1} = V^{N}_{OF_2} = -V^{W}_{OF_1} = -V^{S}_{OF_2} = V^{E}_{F_1I} = V^{N}_{F_2I} = -V^{W}_{F_1I} = -V^{S}_{F_2I}$

$V^{E}_{OF_2} = V^{N}_{OF_1} = V^{W}_{OF_2} = V^{S}_{OF_1} = V^{E}_{F_2I} = V^{N}_{F_1I} = V^{W}_{F_2I} = V^{S}_{F_1I} = 0$

$V^{NE}_{OF_1} = -V^{NW}_{OF_1} = -V^{SW}_{OF_1} = V^{SE}_{OF_1},\ V^{NE}_{OF_2} = V^{NW}_{OF_2} = -V^{SW}_{OF_2} = -V^{SE}_{OF_2}$

$V^{NE}_{F_1I} = -V^{NW}_{F_1I} = -V^{SW}_{F_1I} = V^{SE}_{F_1I}\ ,\ V^{NE}_{F_2I} = V^{NW}_{F_2I} = -V^{SW}_{F_2I} = -V^{SE}_{F_2I}$

$V^{E}_{OS} = -V^{N}_{OS} = V^{W}_{OS} = -V^{S}_{OS}\ ,\ V^{NE}_{OD} = -V^{NW}_{OD} = V^{SW}_{OD} = -V^{SE}_{OD}$

$V^{E}_{SI} = -V^{N}_{SI} = V^{W}_{SI} = -V^{S}_{SI}\ ,\ V^{NE}_{DI} = -V^{NW}_{DI} = V^{SW}_{DI} = -V^{SE}_{DI}$

$V^{\mu}_{OD} = V^{\mu}_{DI} = 0, \qquad \mu \in \{O, E, N, W, S\}$

$V^{\nu}_{OS} = V^{\nu}_{SI} = 0, \qquad \nu \in \{O, NE, NW, SW, SE\}$

$V^{\nu}_{OC} = V^{\nu}_{CI} = 0, \qquad \nu \in \{O, E, N, W, S, NE, NW, SW, SE\}$

Recurrent Connections

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

$V^{E}_{TT} = V^{N}_{TT} = V^{W}_{TT} = V^{S}_{TT}\ ,\ V^{NE}_{SS} = V^{NW}_{SS} = V^{SW}_{SS} = V^{SE}_{SS}$

$V^{E}_{TS} = -V^{N}_{TS} = V^{W}_{TS} = -V^{S}_{TS}\ ,\ V^{E}_{DC} = -V^{N}_{DC} = V^{W}_{DC} = -V^{S}_{DC}$

$V^{NE}_{ST} = -V^{NW}_{ST} = V^{SW}_{ST} = -V^{SE}_{ST}\ ,\ V^{NE}_{CD} = -V^{NW}_{CD} = V^{SW}_{CD} = -V^{SE}_{CD}$

$V^{O}_{TS} = V^{O}_{TF_i} = -V^{O}_{SF_i} = V^{O}_{DC} = V^{O}_{DF_i} = V^{O}_{CF_i} = 0$

$V^{O}_{ST} = V^{O}_{F_iT} = -V^{O}_{F_iS} = V^{O}_{CD} = V^{O}_{F_iD} = V^{O}_{F_iC} = 0$

$V^{\mu}_{DD} = V^{\mu}_{CC} = 0, \qquad \mu \in \{E, N, W, S\}$

$V^{\mu}_{TD} = V^{\mu}_{TC} = V^{\mu}_{SD} = V^{\mu}_{SC} = 0, \qquad \mu \in \{O, E, N, W, S\}$

$V^{\mu}_{DT} = V^{\mu}_{CT} = V^{\mu}_{DS} = V^{\mu}_{CS} = 0, \qquad \mu \in \{O, E, N, W, S\}$

$V^{E}_{SF_1} = -V^{N}_{SF_2} = -V^{W}_{SF_1} = V^{S}_{SF_2},\ V^{E}_{TF_1} = V^{N}_{TF_2} = -V^{W}_{TF_1} = -V^{S}_{TF_2}$

$V^{E}_{F_1S} = -V^{N}_{F_2S} = -V^{W}_{F_1S} = V^{S}_{F_2S},\ V^{E}_{F_1T} = V^{N}_{F_2T} = -V^{W}_{F_1T} = -V^{S}_{F_2T}$

$V^{E}_{SF_2} = V^{N}_{SF_1} = V^{W}_{SF_2} = V^{S}_{SF_1},\ V^{E}_{TF_2} = V^{N}_{TF_1} = V^{W}_{TF_2} = V^{S}_{TF_1} = 0$

$V^{E}_{F_2S} = V^{N}_{F_1S} = V^{W}_{F_2S} = V^{S}_{F_1S},\ V^{E}_{F_2T} = V^{N}_{F_1T} = V^{W}_{F_2T} = V^{S}_{F_1T} = 0$

$V^{E}_{DF_2} = V^{N}_{DF_1} = -V^{W}_{DF_2} = -V^{S}_{DF_1},\ V^{E}_{CF_2} = -V^{N}_{CF_1} = -V^{W}_{CF_2} = V^{S}_{CF_1}$

$V^{E}_{F_2D} = V^{N}_{F_1D} = -V^{W}_{F_2D} = -V^{S}_{F_1D},\ V^{E}_{F_2C} = -V^{N}_{F_1C} = -V^{W}_{F_2C} = V^{S}_{F_1C}$

$V^{E}_{DF_1} = V^{N}_{DF_2} = V^{W}_{DF_1} = V^{S}_{DF_2}\ ,\ V^{E}_{CF_1} = V^{N}_{CF_2} = V^{W}_{CF_1} = V^{S}_{CF_2} = 0$

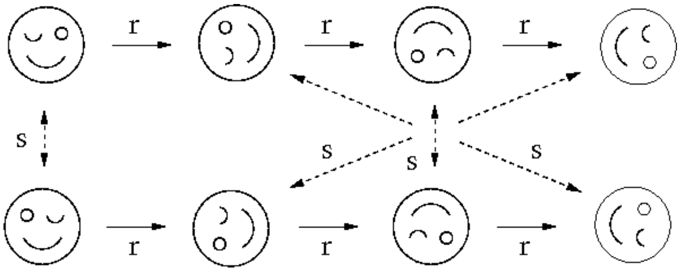
$V^{E}_{F_1D} = V^{N}_{F_2D} = V^{W}_{F_1D} = V^{S}_{F_2D}\ ,\ V^{E}_{F_1C} = V^{N}_{F_2C} = V^{W}_{F_1C} = V^{S}_{F_2C} = 0$

$V^{O}_{F_1F_1} = V^{O}_{F_2F_2} = 0$

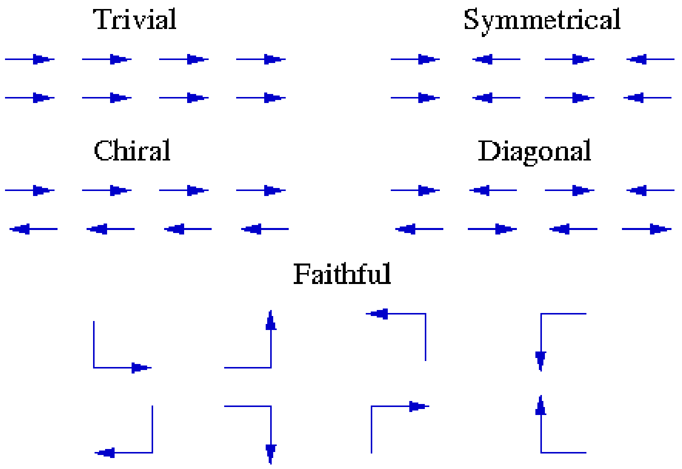
$V^{E}_{F_1F_1} = V^{N}_{F_2F_2} = V^{W}_{F_1F_1} = V^{S}_{F_2F_2}\ ,\ V^{E}_{F_2F_2} = V^{N}_{F_1F_1} = V^{W}_{F_2F_2} = V^{S}_{F_1F_1} = 0$

$V^{\mu}_{F_1F_2} = V^{\mu}_{F_2F_1} = 0, \qquad \mu \in \{O, E, N, W, S\}$

C. **The dihedral group D_8 with generators *r, s***



D **The five irreducible representations of D_8**



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

[5] A. Blair , and G. Li, “Training of Recurrent internal Symmetry Networks by Backpropagation”, IEEE Trans on Neural Networks, Proceedings of the International Joint Conference, 2009. Unpublished

[6] M.Boden, “a guide to recurrent neural network and backpropagation”, Dallas projecst,SICS Technical Report T2002:03, SICS, 2002

[7] S. Lawrence, C. Lee.Giles, and Ah. Chung.Tsoi, “What size of Neural Network gives Optimal generalization? Convergence Properties of Backpropagation”, Technical Report, UMIACS-TR-96-22. and CS-TR-3617. 1996.

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

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

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

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

[12] J. Pollack, “The Induction of Dynamical Recognizers”, Machine Learning 7, pp. 227-252, 1991.