CHARACTER RECOGNITION USING A NEOCOGNITRON

Nicholas J. Conn

Electrical and Microelectronic Engineering Department, Rochester Institute of Technology, Rochester, NY 14623, USA - 2012
nxc9827@rit.edu

ABSTRACT

Artificial neural networks have been used for many years to classify objects and recognize patterns. This paper investigates the use of a self-organizing neural network, called the neocognitron, for use in character recognition. Character recognition requires a system which must succeed despite variations in the character and spatial shifts. For this study, a neocognitron will be created using Java due to the object oriented nature of an artificial neural network. This network will be trained to recognize five characters and then tested against the same characters with noise, with different scales, and with spatial shifts. The neocognitron performs well and is able to recognize characters with reasonable amounts of shift and variation.

Index Terms— Neocognitron, Neural Network, Character Recognition, Pattern Recognition, Artificial Neural Networks, Java, Self-Organization, Unsupervised Learning

1. INTRODUCTION

The neocognitron is a self-organizing neural network which excels at visual pattern recognition. In order to perform character recognition, the methodology must be able to handle variations in the signal. Typically, written characters have not only spatial variations for each user, but also vary greatly in appearance between different individuals.

This problem drives the need for a robust character recognition system that can handle these variations. The neocognitron provides a platform for character recognition which is resilient to changes in character appearance and spatial location. The neocognitron is a self-organizing neural network; this means that the features which are extracted are determined during training.

By using a hierarchal system in which features are extracted within the first few layers, the later layers can provide the character recognition [2]. Such a system is able to be trained to recognize many different types of characters.

Many previous attempts at designing a neural network for character recognition have failed to recognize characters that were distorted in shape, or contain shifts in position [1]. Additionally, the neocognitron was designed to mimic the known functionality of the human brain in how it recognizes some visual patterns. The structure of this network is suggested by the visual nervous system of vertebrate [1]. While the original paper from Fukushima provides initial values for most of the constants [1], and the Zhengjun investigates the optimal values for many of the weights [3], initial values for some of the weights are unknown.

This paper will investigate the ability of the Neocognitron to recognize characters. The mathematics and structure of the neural network will be discussed in Section 2, and then a test dataset will be proposed. Finally, the resulting neural network will be tested with the test dataset; this is shown in Section 3. Conclusions will be discussed in Section 4.

2. PROPOSED APPROACH

The complex structure of the neocognitron allows features to be extracted from an image regardless of shifts in position. This structure is defined by the organization of the weights and connections of the network. In order to better visualize the structure of the neocognitron, first the layout of the network and the terminology used must be understood.

The input is low resolution square image, for this implementation it will be a 16 by 16 image of a given character. Each layer consists of two "sub-layers"; for example, Layer 1 consists of an s-layer and a c-layer. Within each layer, there are multiple s-planes or c-planes. A flowchart of the high level structure can be seen in Figure 1.

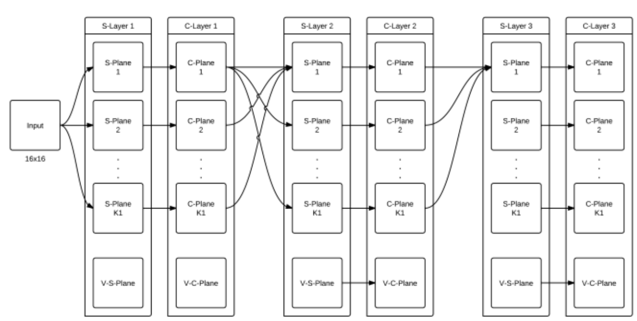

Figure 1: Top level block diagram of the Neocognitron

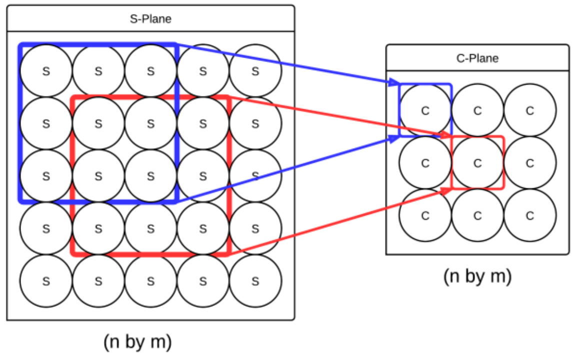

Figure 2: Organization of each plane showing the inter-layer connection window.

Each plane of cells within every layer is a matrix of neurons; the type of neuron depends on which type of layer it is in. The size of the plane is typically square, as shown in Figure 2, where *n* is equal to *m* (5 by 5). Every plane in a given s-layer or c-layer is the same size. Though some connections are shown in Figure 1, not all of them are drawn. Each c-plane is connected only to the preceding s-plane; each s-plane is connected to every preceding c-plane.

Within each layer there also exists a single v-plane, regardless of how many s-planes or c-planes there are. The purpose of each v-cell is to provide an inhibitory response to an s-sell or c-cell in every plane. Due to the fact that there is only one v-plane per layer, every location in each s-plane or c-plane receives the same inhibitory input. The connection of each cell can be seen in Figure 3.

For this implementation, three layers will be used with the size of each layer shown in Figure 1. It is important to note how the last c-layer has a plane size of 1 by 1. After training is completed, the network will propagated a given input; the result is a large output from single plane in the last layer, which consists of a single cell. Thus there should be at least as many planes in the final c-column as there are characters to be recognized.

Before the structure will be defined further, the equations which characterize each cell type will be given. This mathematical foundation will provide a basis for seeing how each layer connects to other layers.

$$u_{Sl}(k_l, \boldsymbol{n}) = r_l \cdot \varphi\left[\frac{1+\sum_{k_{l-1}=1}^{K_{l-1}} \sum_{v \in S_l} a_l(k_{l-1}, \boldsymbol{v}, k_l) \cdot u_{Cl-1}(k_{l-1}, \boldsymbol{n}+\boldsymbol{v})}{1+\frac{2r_l}{1+r_l} \cdot b_l(k_l) \cdot v_{Cl-1}(\boldsymbol{n})} - 1\right] \quad (1)$$

$$\varphi[x] = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases} \quad (2)$$

The above two equations represent the output of an s-cell. The output, $u_{Sl}(k_l,\boldsymbol{n})$, is the output from the *l*-th layer, from the plane k_l at the location $\boldsymbol{n}$; where $\boldsymbol{n}$ is a two dimensional vector, which represents the location of a cell in a specific plane. This equation depends on four values, some of which are vectors and matrix values.

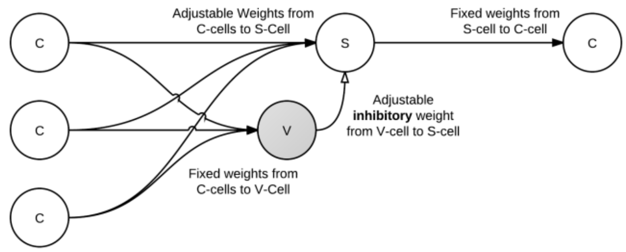

Figure 3: Connections between each type of cell; shows inhibitory response from v-cell.

The input signal, $u_{Cl-1}(k_{l-1},\boldsymbol{n}+\boldsymbol{v})$, represents the signal coming from the previous layer, $_{Cl-1}$.The specific input is different for every plane in the input layer, k_{l-1}. The input also depends on the location $\boldsymbol{n}$, in that specific plane. The range of the inputs used for this cell is determined by a window of cells, S_l, which is centered around a specific point, $\boldsymbol{n}$. This can be visualized in Figure 2, each input windows box, S_l, is centered around a different.

The input weights $a_l(k_{l-1},\boldsymbol{v}, k_l)$ change from plane to plane, they also change depending on which plane the input is coming from in the previous layer. It is important to note that the input weights do not depend on the specific location within the plane. As such, within each plane the weights are the same for any given location; this is what allows features to be extracted regardless of spatial distribution.

It is also important to note that this equation does not allow negative values. Equation 2 forces the output to zeros for any negative output values. Additionally, r_l is used to scale the output when positive. There is a different r_l value for each layer. The final value which the output of an s-cell relies on is from the v-cell plane. There is only one v-cell plane per s-sell layer; as such the v-cell output value depends only on the location in the plane, $\boldsymbol{n}$.

The v-cell output is then weighted by a single value $b_l(k_l)$; this value is different for each plane within a given layer, but is not dependent on the location. Both the $b_l(k_l)$ weights and the $a_l(k_{l-1},\boldsymbol{v}, k_l)$ weights change as the network is trained. These values are what dictate how well the network extracts and recognizes certain patterns.

$$v_{Cl-1}(\boldsymbol{n}) = \sqrt{\sum_{k_{l-1}=1}^{K_{l-1}} \sum_{v \in S_l} c_{l-1}(\boldsymbol{v}) \cdot u^2_{Cl-1}(k_{l-1}, \boldsymbol{n}+\boldsymbol{v})} \quad (3)$$

The above equation shows how the output of each v-cell in the v-plane is determined. This equation is very similar to the numerator in equation 1. The weight matrix, $c_{l-1}(\boldsymbol{v})$, is multiplied by the output squared from the previous c-layer, u_{Cl-1}. The weight matrix $c_{l-1}(v)$ is dependent only on the window. They are the same for every input plane and every location in the single v-plane.

$$\sum_{k_{l-1=1}}^{K_{l-1}} \sum_{v \in S_l} c_{l-1}(\boldsymbol{v}) = 1 \quad (4)$$

This weighting matrix does differ from the other weights in this network due the fact that the summation of the entire matrix, for every plane in the previous layer, must be less than 1. Since the output from each c-cell will always be less than 1, which is shown further on in this section, the output of each v-cell will also be less than one when the above equation is satisfied. Additionally, the output of the v-cell is in essence, the weighted root mean square (RMS) of the previous c-layer's output for a specific location $\boldsymbol{n}$.

The initial values for each weight will be discussed in Section 3, Results. Only the weight matrices $a_l(k_{l-1},\boldsymbol{v}, k_l)$ and $b_l(k_l)$ are updated during training; the $c_{l-1}(\boldsymbol{v})$ weights and the $d_l(\boldsymbol{v})$ weights are not, they remain constant once initialized. This concludes the mathematics needed for implementing the s-layer.

For every s-layer, there is a matching c-layer which contains the same number of planes. Each plane in the c-layer receives inputs from only a single plane in the previous s-layer. This differs from the s-planes which receive their inputs from every plane in the preceding c-layer.

$$u_{Cl}(k_l, \boldsymbol{n}) = \psi\left[\frac{1+\sum_{v\in D_l} d_l(\boldsymbol{v})\cdot u_{Sl}(k_l,\boldsymbol{n}+\boldsymbol{v})}{1+v_{Sl}(\boldsymbol{n})} - 1\right] \quad (5)$$

$$\psi[x] = \varphi\left[\frac{x}{\alpha+x}\right] \quad (6)$$

Equations 5 and 6 show how the output of each c-cell in a c-layer is mathematically determined. The output from the previous s-layer, $u_{Sl}(k_l,\boldsymbol{n}+\boldsymbol{v})$, is multiplied by a weight matrix d. Each c-cell in a given plane receives an input only from the s-cells in the previous layer in the same plane. As such, there is no summation across multiple planes for a specific position.

The weight matrix $d(\boldsymbol{v})$, is only dependent on the window contained in D_l, and not which plane the c-cell is in nor the location within the specific c-plane. Therefore, there is a single set of weights for each c-layer; compared to s-layers which contain a different set of weights for plane of inputs and outputs.

As with the s-cell, each c-cell receives an inhibitory input from a matching v-cell. For each c-layer, there exists only one v-plane. For every plane, there is a specific v-cell for each location; the output of a v-cell is determined using equation 7.

$$v_{Sl}(\boldsymbol{n}) = \frac{1}{K_l}\cdot \sum_{k_{l=1}}^{K_l}\sum_{v\in D_l} d_l(\boldsymbol{v})\cdot u_{Sl}(k_l,\boldsymbol{n}+\boldsymbol{v}) \quad (7)$$

The same weighting value is used for the v-cells in the c-layer as for each c-cell, $d_l(\boldsymbol{v})$. As previously mentioned, the initial values for each $d_l(\boldsymbol{v})$ will be discussed in the results section, Section 3. Unlike the c-planes in the c-layer, the v-plane within the c-layer receives inputs from each plane in the preceding s-layer. This concludes the mathematics needed to propagate a signal through the network.

In order for the network to learn certain patterns, and to self-organize, it must be trained. The first version of the neocognitron proposed by Fukushima uses unsupervised training [1]. Only two weights are modified during training, $a_l(k_{l-1},\boldsymbol{v}, k_l)$ and $b_l(k_l)$.

In order to train the network representative cells for each layer must be determined each time an image is propagated. This methodology of training is a "winner takes all" approach. The cells with the highest output are reinforced, while the other cells are left alone. This approach allows each cell plane to self-organize so that it will recognize only one specific feature from its inputs.

Since only the s-layer weights will be trained, the representative cells will only be determined within each s-layer. Each s-plane in the layer to be trained is stacked on top of one another, like a deck of cards. A stack of s-cells for each position now exist. By using a window around each stack of s-cells, similar to that shown in Figure 2, an s-column is created. As an example, if the window being used is 3 pixels by 3 pixels and there are 8 planes, each s-column "layer" will be 3 by 3 s-cells with a total of 8 "layers".

This s-column is generated for every pixel location; as such a given s-cell can be part of multiple s-columns. Within each s-column, the plane and the location of the s-cell with the largest output are chosen as representative. Since there are multiple s-columns it is possible that more than one representative cell is chosen within a given plane; in this case, the one with the largest output is chosen so that no more than one cell is chosen per plane.

For each plane which contains a representative cell, $\hat{k}_l$, the *a* and *b* weights are increased the amount dictated by equations 8 and 9. The amount each cell is increased is determined by using the input at position $\hat{\boldsymbol{n}}$. By increasing these values, each plane will recognize a single feature.

$$\Delta a_l\left(k_{l-1}, \boldsymbol{v}, \hat{k}_l\right) = q_l\cdot c_{l-1}(\boldsymbol{v})\cdot u_{Cl-1}(k_{l-1}, \hat{\boldsymbol{n}}+\boldsymbol{v}) \quad (8)$$

$$\Delta b_l\left(\hat{k}_l\right) = \frac{q_l}{2}\cdot v_{Cl-1}(\hat{\boldsymbol{n}}) \quad (9)$$

For every character which is propagated by the network during training, every s-layer's weights are updated using the method described above. For both equation 8 and 9, the constant q_l dictates how quickly the network will become trained. A different q_l value is used for each s-layer. This concludes the theoretical foundation needed to implement a neocognitron.

3. RESULTS

Using Java, a neocognitron was built. This neocognitron can receive an input image of 16 pixels by 16 pixels. There are three total layers, as shown in Figure 1; each layer contains the same number of planes. The first s-layer has a cell resolution of 16 by 16 pixels and the first c-layer has a resolution of 10 by 10 pixels. The second s-layer has a resolution of 8 by 8 pixels and the second c-layer has a resolution of 6 by 6 pixels. The final s-layer has a resolution of 2 by 2 pixels, and the final c-layer contains only one pixel per plane.

The weighting constants d, and c, were determined using equations 10 and 11 [3]. It is important that both weights be monotonically decreasing; by using values for both δ and γ which are less than one, this requirement will be met.

$$c_l(\boldsymbol{v}) = \gamma_l^{|\boldsymbol{v}|} \tag{10}$$

$$d_l(v) = \bar{\delta}_l \cdot \delta_l^{|\boldsymbol{v}|} \tag{11}$$

The number of planes per layer and all the constants used for the final neocognitron were determined experimentally by cycling through a range of possibilities in order to determine the best values. The optimal values which were determined using this approach are shown in Figure 4.

All b weights are initialized to zero, regardless of layer; and all a weights are initialized to a random value no greater than 0.5. The constant α was set to .478 and the number of planes in each layer is 16. Now that all the needed initial values are determined, the neocognitron can be trained using a set of test characters.

The characters shown in Figure 5 were used for testing and verification of the neocognitron. Since the final recognition rate is determined by the initial values of the randomly generated $a_l(k_{l-1},\boldsymbol{v}, k_l)$ weights, many neocognitron are created, trained, and tested; the one containing the highest recognition rate is saved for further use.

The fourth number of each sequence shown in Figure 5 is used to train the neocognitron; all of the numbers are used for determining the error rate of the network. A neocognitron created with the values shown in Figure 4 was able to realize a 73.3% recognition rate. Out of the twenty characters in figure 5, the network was unable to recognize four characters. Three of the four unrecognized characters was the number one; while the last unrecognized character was the first number four shown in Figure 5.

Due to the fact that the number one is a very simple character and that it does not have many features, explains why it was not recognized by the neocognitron. Despite many differences between the verification characters and the training characters, the neocognitron performed very well. Despite the additional pixels and changes to the size and shape of each number, the neocognitron successfully recognized a large percentage of the verification characters.

	Layer 1	Layer 2	Layer 3
r_l	4.81	1.44	2.57
q_l	0.20	9.60	13.94
γ_l	0.11	0.42	0.06
δ_l	0.49	0.87	0.52
$\bar{\delta}_l$	0.12	.006	0.78

Figure 4: Constant values used for each layer

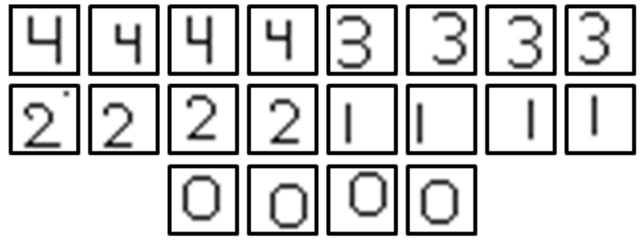

Figure 5: Characters used for training and verification

The values shown in Figure 4 match many of the values used in Fukushima's original network [1]. The first layer's r_l value is the largest, the q_l value increases in each layer, and the constant α is approximately .5. Additionally, the number of planes is at least twice as large as the number of characters the network can recognize.

This neocognitron can be trained to recognize up to sixteen different characters. If more than sixteen characters must be recognized, the number of planes will need to also be increased. Overall, the network was very successful in recognize numbers zero through four despite the fact that the characters were scaled and shifted spatially.

4. CONCLUSIONS

Intuitively, the neocognitron is perfectly designed for visual pattern recognition. The structure and theory behind the network implies a robust method for character recognition. Despite this, the implementation of the neocognitron proposed by Fukushima is not straight forward [1]. There are many values which need to be set perfectly for pattern recognition to be successful. By testing the network with many variations on these values, an optimal set of parameters have been found. Once such values were found, the trained network was able to recognize a set of test characters, which was different from the training set, with a 73.3% success rate. Further studies could analyze the networks ability to recognize the entire English alphabet. To conclude, the neocognitron can be successfully trained to recognize characters despite shifts in position.

5. REFERENCES

[1] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," Biological Cybernetics, vol. 36, pp. 193-202, 1980.

[2] C. N. S. Ganesh Murthy and Y. V. Venkatesh, "Modified neocognitron for improved 2-D pattern recognition," Vision, Image and Signal Processing, IEE Proceedings -, vol. 143, pp. 31-40, 1996.

[3] Zhengjun Pan; Sabisch, T.; Adams, R.; Bolouri, H.; , "Staged training of Neocognitron by evolutionary algorithms," Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on , vol.3, no., pp.3 vol. (xxxvii+2348), 1999