A Scheme for Implementation of Neural Networks
With Replicated Receptive Fields

Michael Chuang

Massachusetts Institute of Technology

Abstract

In this paper we show how neural networks with local receptive fields and replicated weights can be mapped efficiently onto a CCD parallel processing architecture. Implementation of the neocognitron, a neural network for feature extraction and classification, on the CCD architecture was simulated. A modified training procedure for the neocognitron that improves its ability to extract features when using the CCD architecture is presented.

Introduction

A multilayer neural network characterized by local connectivity patterns and groups of nodes constrained to have the same weights on their input lines can be described as a hierarchical collection of spatially replicated feature detectors. Such networks have been applied successfully to tasks ranging from handwritten character recognition to phoneme extraction in speech preprocessing. In the latter case, delayed copies of the input are presented to the network in addition to the actual input, and the weights should be thought of as temporally replicated. Neural networks with local connections and groups of identical weights will be referred to in this paper as replicated-weight neural networks (RWNN).

RWNNs have generally been used to classify small input images, but the replicated-weight structure can also be applied to networks for processing large, complex images, where the desired output consists of one or more feature maps of the input image. A charge-coupled device (CCD) architecture that performs high-speed inner products of windowed portions of an input image and stored weight templates is well suited for implementation of RWNNs. This paper describes the CCD architecture and how an RWNN algorithm can be mapped onto that architecture. Implementation of a specific RWNN, the neocognitron, has been simulated, and a modification to the neocognitron training procedure that ameliorates the effects of weight quantization, which would be imposed by use of the CCD architecture, is presented.

CCD Image Processor Architecture

A CCD architecture that can be used for image processing tasks, such as two-dimensional FIR filtering or motion-compensated image processing using region matching techniques, has been developed at Lincoln Laboratory by Chiang. This architecture is also suited for implementation of multilayer feature map-generating RWNNs. The building blocks of the architecture consist of a CCD tapped delay line for holding and shifting input pixels or node values, digital CCD shift registers for on-chip storage of (fully programmable) filter coefficients or neural network connection weights, and multiplying digital-to-analog converters (MDACs) for computing the inner products of stored weights and input values. Devices built using the CCD image processor architecture have simple i/o requirements, are compact, and consume little power while offering high throughput computation.

Perhaps the best way of describing the CCD image processing architecture in greater detail is to refer to a specific example: an "image feature extractor" (IFE) was recently reported by Chiang and LaFranchi. The IFE is designed to process an input image 128 columns wide. The number of rows in the input is essentially arbitrary, though we generally will assume, when discussing the time required to process "an image" for example, that the input is 128x128 pixels. Conceptually, the IFE slides a 7x7 window over the input image. At each window position, inner products of the stored weight templates (of which there may be up to 20) and the windowed portion of the input are computed. The input pixels are read into the IFE row-by-row and the task of selecting the appropriate pixels for windowing is performed on-chip; memory access patterns are kept simple.

The IFE contains a 775-stage tapped delay line that holds six 128-pixel lines of the input image plus an additional seven pixels. The input values are represented by analog charge packets. Taps placed at appropriate points along the delay line allow selected pixels or node values to be sensed nondestructively so that the values read from the one-dimensional line correspond to those that would be covered by a two-dimensional 7x7 window. This is shown in figure 1. (The actual tapped delay line is implemented as a U-shaped structure, but is depicted in figure 1 as a serpentine structure for heuristic purposes.) The values read from the tapped delay line are multiplied in parallel by 8-bit digital weights using MDACs. The resulting charge domain partial products are summed in a common output node; this allows a complete inner product computation to be performed on each clock. Digital cyclic shift registers are used to store the weights so that each set of weights can be applied to the windowed pixels before shifting the input data. Shifting the contents of the tapped delay line by one position corresponds to moving the window over the image by one pixel.

Although the operation of the IFE is most easily visualized as sliding a window over an input image, in reality the pixels of the image are "pumped through" the device row-by-row. It should also be noted that the organization of the IFE permits input data to be loaded at a rate one-twentieth that of the internal device clock so that high computation rates can be achieved despite relaxed input rates. Clocked at 20 MHz, the image feature extractor device can process a 128x128-pixel input image to produce twenty filtered output images or feature maps in 16.4 msec.

Feature Extraction and Classification Using RWNNs

Neural networks with replicated weights can be used as classifiers for small images such as preprocessed (e.g. thinned, segmented, size-normalized) handwritten characters or to generate feature maps from large, complex images. The basic
The idea in both cases is to use the first layer of the network to break up the image into a collection of simple features, such as oriented line segments. Higher layers put the pieces back together into successively more complex features. If the network is to be used as a classifier, then the output layer consists of a small number of nodes where each node corresponds to a single learned category. A handwritten-digit classifier, for example, would have ten output nodes. The output of a feature map-generating RWNN consists of one or more arrays that are feature maps of the input image.

To give a more concrete explanation of the feature extraction/synthesis process, consider a multilayer RWNN that accepts an N×N-pixel image as input. Each node in the first layer of the network is connected to a small group of neighboring pixels. All first-layer nodes have the same weights, but each node is connected to a different set of input pixels. The pixels to which a node is connected constitute that node's "receptive field." Receptive fields may overlap, so that a given pixel can be connected to several first-layer nodes. Each node is a feature detector; a node with a high activation indicates that the feature of interest is present in its receptive field. If the nodes of the first layer are arranged in a rectangular array, then the first layer is quite literally a feature map of the input image. The second layer receives the activation values of the first layer as input, and it generates a map of more complicated features. This process continues on up to the output layer of the network. In general, the number of nodes in each layer decreases as the layer number increases. This is because a high-layer node has an effective receptive field (the union of the receptive fields of the lower-layer nodes from which it receives input) that covers a much larger portion of the input image than a low-layer node.

In the network described above, each layer looks for only a single feature, but we would like a network that can search for multiple input features. One method of accomplishing this is to partition the nodes of each layer into smaller groups or "node-planes" where each layer has as many independent sets of weights as there are node-planes and all the nodes in a given node-plane have the same weight values. Each node-plane of the first layer covers the entire input, so the first layer generates as many feature maps of the input image as there are node-planes. This allows the first layer to detect multiple features. The second-layer node-planes are connected to one or more first-layer node-planes, so that the second layer can detect and form maps of more complex features from the first-layer representation of the input image.

RWNNs, such as a handwritten digit classifier developed at AT&T, have been applied successfully to classification of small input images. In the case of large, complex images, the task of an RWNN should be the production of a number of feature maps of the input, so that small (relative to the size of the image) patterns or features can not only be identified but their position in the input image localized. If the task is classification of a complex image, an RWNN is unlikely to be the best method of performing the final classification, since the number of layers that would be required to distill the information in the input image down to single nodes activated by the presence of a particular pattern is prohibitively large. The final classification may of course be performed based on feature maps generated by an RWNN. In addition to this very practical issue, there is another motivation for using techniques other than RWNNs for classification of large images. At this point it might be useful to draw a biological analogy, since the organization of an RWNN and the hypercolumn model of the early visual system have many similarities. The early visual system—the pathway from retina to striate cortex—contains cells that respond preferentially to specific simple features, much as the nodes in an RWNN do. The striate cortex maps the entire visual field, but the actual process of recognizing or classifying what is in the visual field is
presumably performed in higher areas of the brain. An RNWNN-like structure performs the task of generating feature maps admirably, but generally is not the appropriate mechanism for classifying large images.

**Implementation of Replicated-Weight Networks**

The description of the first level of an RNWNN as a collection of spatially replicated feature detectors with overlapping receptive fields indicates that this structure is amenable to implementation using the CCD architecture. The mapping of the first layer onto an IFE device (we now use "IFE" as a generic term for both the specific device described above as well as similar devices built in the CCD architecture) is fairly straightforward. Suppose we have an RNWNN that accepts an N×N-pixel image as input. Let the first layer of the network be divided into k node-planes with nxn receptive fields. This implies that the first layer will have k sets of weights. These weights can be stored on-chip in n² k-stage digital shift registers. An [N(n×n)+n]-stage CCD delay line will serve to hold and shift input pixels. Taps are placed at appropriate points along the delay line to simulate the window corresponding to the receptive fields of the nodes. At each window position, k inner products of the stored weights and the windowed pixels are performed. Each inner product, which may be passed through a nonlinear output function that can also be implemented in order, is a node output. Figure 2 depicts the node-planes of the first layer as k feature maps generated from the input. With the window initially over the upper lefthand corner of the input image, node outputs y₁₁,y₂₁,...,yₙₓ_n are generated in order.

![Figure 2. Generation of Feature Maps Using an IFE Device](image)

The mapping of higher layers onto the architecture is somewhat more involved. Suppose that the second layer of our network is also partitioned into k node-planes and that each second-layer node has a compound receptive field that covers the same coordinates in every first-layer feature map. Layers 1 and 2 can be implemented using k+1 IFE devices as shown in figure 3. A single device, labelled IFE(1,1), performs the first-layer computations, both the dot product and the output nonlinearity. The second-layer computations are divided up among the k devices labelled IFE(2,1),...,IFE(2,k). Each output value from IFE(1,1) is sent to exactly one of the second-layer devices, and each second-layer device receives an output once every k clocks. For example, if the output produced by IFE(1,1) is sent to IFE(2,1) at time t₀, then the next output point is sent to IFE(2,2) at time t₀ + Δ, and so forth. IFE(2,k) receives a value at time t₀ + (k-1)Δ, and then IFE(2,1) again receives a value a time t₁ = t₀ + kΔ. (The times t₀, t₁,... are reference times based on the times input pixels are loaded and indicate shifts of the window position. The internal IFE device clocks run k-times faster than the rate at which new data points are loaded, so the Δ is an offset from each reference time, where Δ = (t₁ - t₀)/k.)

The distribution of outputs from IFE(1,1) to the second-layer devices can also be explained using the description of the first-layer node-planes as feature maps of the input image. Recall that an IFE device generates k feature maps of its input and that the pixels of the k feature maps are output in interleaved fashion. That is, the IFE device will output pixel (i,j) of feature map 1, then pixel (i,j) of feature map 2, and so forth through pixel (i,j) of feature map k, after which the next output is pixel (i+1,j) of feature map 1. Physically, the delay line of IFE(2,k) holds the pixels of the kth first-layer feature map.

Outputs of the second layer are not generated directly by the second-layer IFEs since each node of the second layer has a receptive field over every first-layer node-plane (if we do not want any connections to particular node-planes then the appropriate stored weights are set to zero). The output of IFE(2,1) at time tₜ is only a part of the complete output of the first node-plane of layer 2. To this output must be added the output of IFE(2,2) at time tₜ+Δ on through the output of IFE(2,k) at time tₜ+(k-1)Δ. This can be achieved by accumulating these partial outputs in a common well before evaluating the node output. In figure 3 these wells are shown by the circled summation symbols. A similar scheme is used if higher layers are required. After an initial delay (required to load the tapped delay lines) equal to the length of the delay line multiplied by the load-data period, every processor is busy performing useful computations all of the time.

![Figure 3. Implementation of Multilayer Network Using IFE Devices](image)

The Neocognitron

The neocognitron is a multilayer feed-forward RNWNN for feature extraction and pattern recognition that was first proposed by Fukushima in the early 1980's. The neocognitron contains three types of nodes, s-cells for feature extraction, c-cells that compensate for shifts of features between levels, and v-cells that generate inhibitory signals to reduce spurious activation of s-cells. Each layer of the neocognitron contains several node-planes that consist of only s-cells, an equal number of c-cell node-planes and exactly one node-plane of v-cells. A node-plane of s-cells, for example, is called an s-plane. All the cells in a given s-plane have identical weights on their input lines but have receptive fields covering different portions of the input. An s-plane is thus a feature map of the input. For each s-plane there is a corresponding c-plane. Each c-cell has a receptive field on its s-plane; if any of the s-cells in its receptive field are activated then the c-cell itself is activated. A c-plane is a feature map that is indifferent to small shifts of features. A schematic of a three-layer neocognitron is shown in figure 4. V-planes are not shown in the figure, and many connections are omitted to avoid cluttering the diagram.

The computational functions of s- and c-cells are given by equations (1) and (2) below. In the notation used, λ refers to the layer to which a template or cell belongs, so that sj(λ,m,n) refers to the s-cell at position (m,n) in the kth s-plane of layer λ.

---

1991 VLSITSA 71
The parameters $\sigma_a$ and $\sigma_s$ are constants that determine when the s-cell characteristic saturates and the efficiency of the inhibitory input from the v-cell at position $(m,n)$, respectively. (In a given layer, the s-planes and the v-plane have the same dimensions, while c-planes are slightly smaller.) The quantity $b_h(k)\geq 0$ is the strength of the inhibitory connection to the v-cell. The excitatory weights are given by $a_{L(k,K,i,j)}$; $k$ and $K$ indicate the s- and c-planes, respectively, between which the weight template serves a connection, and $(i,j)$ are spatial coordinates within the template. The expression $d_{L(k,K,i,j)}$ is a weighting function on the node inputs to a c-cell:

$$s_j(k,m,n) = \begin{cases} 0, & z < 0 \\ z, & z \geq 0 \end{cases}$$

$$\sigma_a + \sum_{i=1}^{nJ} \sum_{j=1}^{mJ} d_{L(k,K,i,j)} \cdot c_{L_4}(k,m+i-1,n+j-1)$$

$$y = \sum_{i=1}^{nJ} \sum_{j=1}^{mJ} d_{L(k,K,i,j)} \cdot s_j(k,m+i-1,n+j-1)$$

Figure 4. Schematic of a Three-layer Neocognitron

The neocognitron is trained by modification of the $a_{L(k,K,i,j)}$ and $b_h(k)$ weights. Initially, weights are set to small random values. Inputs are then presented to the untrained network. Some s-cells will produce small output values, while others will produce larger values. Assuming that a large output value means that a cell has something interesting in its receptive field, the s-cell with the largest response in each s-plane has a scaled copy of the windowed input added to its weight template. In this way, the $a_{L(k,K,i,j)}$ templates come to represent features. The $b_h(k)$ are also increased whenever the corresponding $a_{L(k,K,i,j)}$ are updated. This training procedure results in weights that grow so long as training is enabled. The unbounded growth of weights can be troublesome in some circumstances, as will be explained in the following subsection.

As can be seen from equations (1) and (2), the bulk of computation in the neocognitron consists of inner products of windowed portions of cell-planes (or the input image) and stored weight templates. The computational requirements of the network appear to be well matched to the capabilities of IFE devices. What must be determined is the effect, if any, that limited arithmetic precision and quantized weight values have on the performance of the neocognitron. The following describes simulation of a neocognitron in which the inner products are computed using IFE devices.

**Simulation Results**

A three-layer neocognitron with ten s-planes and ten c-planes in each layer was implemented in software on a SPARCstation 1 and trained to recognize the four capital letters A, P, S, and T. The letters were represented as 20x20-pixel binary images. The test set consisted of 80 examples of each letter for a total of 320 test characters. All were drawn by a single person on a 20x20 bitmap. The neocognitron was first implemented using floating-point arithmetic for all calculations. The program was then modified to simulate the effect of quantized weights and low-precision arithmetic. (The weights used in the latter case were quantized copies of weights obtained using floating-point arithmetic in training. No networks were trained using low-precision arithmetic as the neocognitron learning rules require the ability to make very small modifications in the initial passes through the training set. Weight values and arithmetic quantized to step sizes that realistically represent implementation by IFE devices result in over-large jumps in weight values; one set of weights ends up overpowering the other weights early in training, producing a neocognitron that lumps all inputs into one category.) As might be expected, the ability of the trained network to identify characters correctly decreased with arithmetic precision. The results are shown in figure 5. Incidentally, these results do not necessarily indicate how the neocognitron might perform on a "real world" problem since the test set was generated by a single person and included grossly deformed characters with very deliberate writing errors.

The recognition capability of the neocognitrons decreased because quantization clipped large weights and effectively set small weights to zero. Some experiments with deleting a fraction of the weights according to some rule (e.g. get rid of really big weights or get rid of really small weights) and then allocating the available bits of precision to span the range of the remaining weights did not solve the problem. Although approximately half of the learned weights apparently did not contribute to, or detract from, recognition (this was determined by randomly deleting weights), essential weights might be very large or very small. One possible solution appeared to be changing the training procedure to eliminate the unbounded growth of weights. The effect of this modification is described below.

Closer examination of an s-cell reveals that the weight vector $a_{L(k,K,i,j)}$ and an input vector are normalized implicitly by the s-cell during the dot product-and-threshold computation. This process is described in detail elsewhere by Fukushima. In order to eliminate the (numerical range-wasting) normalization during evaluation, a modified training procedure that explicitly normalized the weight vectors during training, after they reached a preselected magnitude, was implemented. The original training procedure rotated weight vectors by increasing the sizes of components as needed, while the new procedure rotates a
weight vector without increasing its magnitude. A network trained using the modified procedure has a slightly lower (approximately three percent reduction) recognition capability when floating-point arithmetic is used, but performs significantly ($\alpha < .025$ for 10 bits, $\alpha < .0005$ for 9, 8, and 7 bits by paired t-test) better when quantized weights and low-precision arithmetic is used. The results are shown below in table 1.

A simulation of implementation of the neocognitron, an RWNN for feature extraction and pattern recognition, using the CCD devices was performed. It was determined that weight quantization and limited arithmetic precision would substantially degrade the performance of the neocognitron. A modified training procedure that reduces quantization effects was proposed and tested. The modified training algorithm, which explicitly normalizes weight vectors during training, produced networks that had significantly better performance than networks trained using the original procedure.

### Reference


### Acknowledgements

This work was sponsored by the Office of Naval Research and the Department of the Air Force.

### Table 1. Percentage of Correctly Recognized Inputs Relative to Network Using Floating-Point Arithmetic

<table>
<thead>
<tr>
<th>Precision (bits)</th>
<th>Original</th>
<th>Modified</th>
</tr>
</thead>
<tbody>
<tr>
<td>floating-point</td>
<td>100.0%</td>
<td>100.0%</td>
</tr>
<tr>
<td>10-bit</td>
<td>97.2%</td>
<td>98.6%</td>
</tr>
<tr>
<td>9-bit</td>
<td>94.3%</td>
<td>96.0%</td>
</tr>
<tr>
<td>8-bit</td>
<td>89.4%</td>
<td>96.0%</td>
</tr>
<tr>
<td>7-bit</td>
<td>74.5%</td>
<td>91.9%</td>
</tr>
</tbody>
</table>

In this paper we have identified a CCD architecture that is particularly well-suited to the computational requirements of neural networks with local connections and replicated weights. We propose that for large, complex images, the proper function of an RWNN is to produce feature maps of the image; an RWNN is not necessarily the appropriate mechanism to perform classification, although it can certainly be a useful component of a classification system. It is shown how RWNNs designed to generate feature maps from large input images can be implemented using the architecture. The processing devices in a CCD implementation of such RWNNs are kept busy performing useful computations virtually 100 percent of the time, apart from the initial period during which the tapped delay lines that hold node values are loaded. Devices built using the CCD architecture are compact, low-power, and perform the multiplications and accumulations of the generic neural network node operation (inner product computation) in parallel.

In this paper we have identified a CCD architecture that is particularly well-suited to the computational requirements of neural networks with local connections and replicated weights. We propose that for large, complex images, the proper function of an RWNN is to produce feature maps of the image; an RWNN is not necessarily the appropriate mechanism to perform classification, although it can certainly be a useful component of a classification system. It is shown how RWNNs designed to generate feature maps from large input images can be implemented using the architecture. The processing devices in a CCD implementation of such RWNNs are kept busy performing useful computations virtually 100 percent of the time, apart from the initial period during which the tapped delay lines that hold node values are loaded. Devices built using the CCD architecture are compact, low-power, and perform the multiplications and accumulations of the generic neural network node operation (inner product computation) in parallel.

### Summary

In this paper we have identified a CCD architecture that is particularly well-suited to the computational requirements of neural networks with local connections and replicated weights. We propose that for large, complex images, the proper function of an RWNN is to produce feature maps of the image; an RWNN is not necessarily the appropriate mechanism to perform classification, although it can certainly be a useful component of a classification system. It is shown how RWNNs designed to generate feature maps from large input images can be implemented using the architecture. The processing devices in a CCD implementation of such RWNNs are kept busy performing useful computations virtually 100 percent of the time, apart from the initial period during which the tapped delay lines that hold node values are loaded. Devices built using the CCD architecture are compact, low-power, and perform the multiplications and accumulations of the generic neural network node operation (inner product computation) in parallel.

A simulation of implementation of the neocognitron, an RWNN for feature extraction and pattern recognition, using the CCD devices was performed. It was determined that weight quantization and limited arithmetic precision would substantially degrade the performance of the neocognitron. A modified training procedure that reduces quantization effects was proposed and tested. The modified training algorithm, which explicitly normalizes weight vectors during training, produced networks that had significantly better performance than networks trained using the original procedure.

### References


### Acknowledgements

This work was sponsored by the Office of Naval Research and the Department of the Air Force.