



SAPIENZA
UNIVERSITÀ DI ROMA

Facoltà di Ingegneria
INGEGNERIA INFORMATICA
(Intelligenza Artificiale)

Corso: Machine Learning

Docente: Prof. LUCA IOCCHI

MULTILAYER FEEDFORWARD NEURAL NETWORKS
(ALPHANUMERIC RECOGNITION)

REDJAN SHABANI
(1013173)

Contents

INTRODUCTION	2
1-MULTILAYER NEURAL NETWORKS.....	2
1.1-Sigmoid Neuron	2
1.2-Neural Layer with Sigmoid Neuron.....	3
1.3-Feedforward Layered Neural Network	4
2-BACKPROPAGATION ALGORITHM	5
2.1 - Gradient Descent Training Rule	5
2.1-Stochastic Gradient Descent & Delta Rule	6
2.2-Backpropagation algorithm	7
3-ALPHANUMERIC RECOGNITION.....	10
3.1-Typescript Recognition	10
3.2-Manuscript Recognition.....	14
References	17

INTRODUCTION

This paper describes the small research done by the author regarding the *character recognition problem*, making use of *artificial neural networks* (ANN). First is presented the mathematical model of ANN with *sigmoid neurons*. Next there is an introduction to the *back-propagation algorithm*, widely used for training ANN-s over a set of examples. Then, is presented the character recognition problem, divided in two macro classes, *typescript recognition* and *manuscript recognition*.

1-MULTILAYER NEURAL NETWORKS

1.1-Sigmoid Neuron

The generic neural function, that relates the unit inputs with the unit output, is defined by:

$$y = f\left(w_0 + \sum_{i=1}^n w_i x_i\right)$$

w_0 is called unit bias. It is possible to consider a bias as part of the weighted sum of inputs by supposing to have a pseudo-input $x_0 = +1$, so we can write:

$$y = f\left(w_0 + \sum_{i=1}^n w_i x_i\right) = f\left(\sum_{i=0}^n w_i x_i\right), \quad (x_0 = +1)$$

A *sigmoid neuron* is characterized by the following threshold function:

$$\sigma_{\tau}(t) = \frac{1}{1 + e^{-\frac{t}{\tau}}}$$

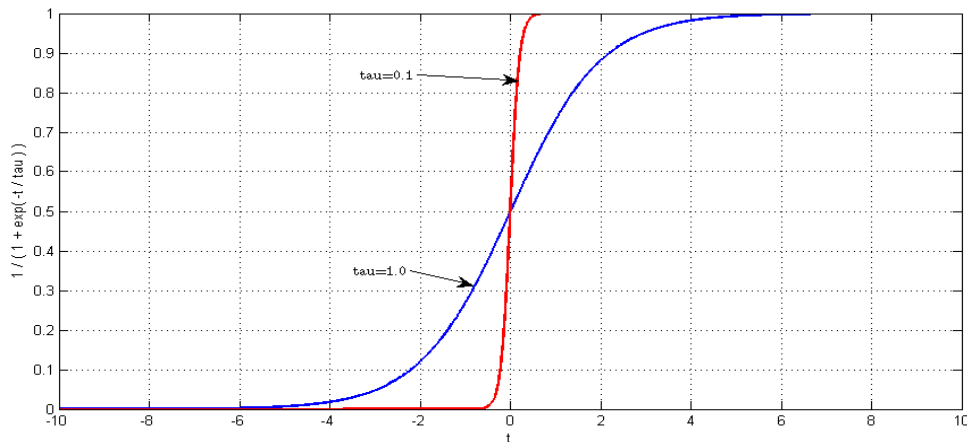


Figure 1-Sigmoid function plot

The neural function for a sigmoid neuron is:

$$y = \sigma_{\tau} \left(\sum_{i=0}^n w_i x_i \right) = \frac{1}{1 + e^{-\frac{\sum_{i=0}^n w_i x_i}{\tau}}}$$

The sigmoid has the characteristic of being continuous function in the entire real domain and so is even its derivative. Sigmoid function can be used for classifying data with a kind of uncertainty in regions where the classes are too near. If we want to make a classical rigid classification we may simply use $\tau = 0$, obtaining the Heaviside¹ threshold function.

Let's see the derivative of the sigmoid for $\tau = 1$:

$$\begin{aligned} \frac{\partial \sigma(t)}{\partial t} &= \frac{\partial}{\partial t} \left(\frac{1}{1 + e^{-t}} \right) = -\frac{1}{(1 + e^{-t})^2} \cdot \frac{\partial}{\partial t} (1 + e^{-t}) = -\frac{1}{(1 + e^{-t})^2} \cdot (-e^{-t}) = \frac{e^{-t}}{(1 + e^{-t})^2} = \frac{(1 + e^{-t}) - 1}{(1 + e^{-t})^2} \\ &= \frac{1}{1 + e^{-t}} - \frac{1}{(1 + e^{-t})^2} = \sigma(t) - [\sigma(t)]^2 \end{aligned}$$

so:

$$\dot{\sigma}(t) = \sigma(t)(1 - \sigma(t))$$

1.2-Neural Layer with Sigmoid Neuron

A neural layer is a structure containing a set of neurons. All neurons in the layer have the same number of inputs. The input size of neurons is called even the inputs size of the layer. The layer captures the input vector and passes it to every unit. The output produced by every unit constitutes the output vector of the layer. As we can see a layer has a multiple output and we can see this structure as the generalization of the neural unit that has a single output.

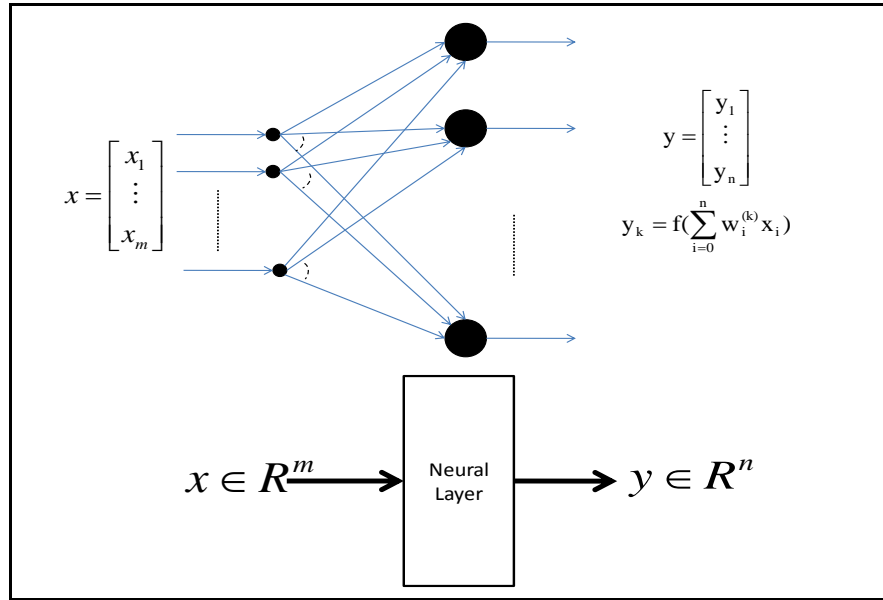


Figure 2-Neural Layer structure.

Let's make some symbolic definitions:

- $\mathbf{x} = [x_1 \quad \dots \quad x_m]^T$: layer input vector

¹ The Heaviside function $H(t) = f(x) = \begin{cases} 1, & t > 0 \\ 0, & t \leq 0 \end{cases}$ is not derivable in $t=0$. The sigmoid function does not have this problem.

- $\mathbf{y} = [y_1 \quad \dots \quad y_n]^T$: layer output vector
- $\mathbf{w}^{(k)} = [w_1^{(k)} \quad \dots \quad w_n^{(k)}]^T$: the weights vector of the k-th unit in the layer
- $\mathbf{W} = [\mathbf{w}^{(1)} \quad \dots \quad \mathbf{w}^{(n)}]$: weights matrix of the layer
- $f(u)$: threshold function

The threshold function $f(u)$ is defined for real argument and returns real values. With a little bit of mathematical abuse, we will extend the argument domain to vectors of real. If $\mathbf{u} = [u_1 \quad \dots \quad u_m]^T$ is a vector of real number than we define the extension of a real function $f: \mathbb{R} \rightarrow \mathbb{R}$ as a function $\mathbf{f}: \mathbb{R}^n \rightarrow \mathbb{R}^n$ so defined:

$$\mathbf{f}(\mathbf{u}) = \mathbf{f}\left(\begin{bmatrix} u_1 \\ \vdots \\ u_m \end{bmatrix}\right) = \begin{bmatrix} f(u_1) \\ \vdots \\ f(u_m) \end{bmatrix}$$

Because we will use sigmoid units we have to obtain the extension of the sigmoid function, which is simply:

$$\sigma\left(\begin{bmatrix} u_1 \\ \vdots \\ u_m \end{bmatrix}\right) = \begin{bmatrix} \sigma(u_1) \\ \vdots \\ \sigma(u_m) \end{bmatrix} = \begin{bmatrix} (1 + e^{-u_1})^{-1} \\ \vdots \\ (1 + e^{-u_m})^{-1} \end{bmatrix}$$

Now we to obtain the layer function that relates inputs and outputs. Before this let's rewrite weighted sum before threshold of neurons in terms of matrix product:

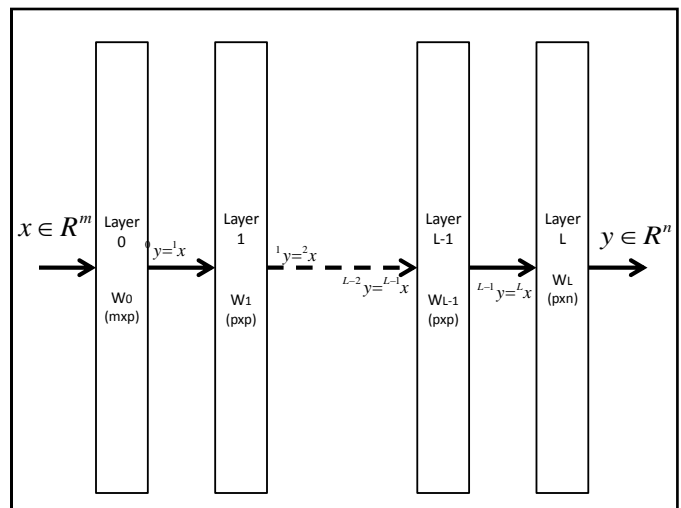
$$y_k = \sum_{i=0}^n w_i^{(k)} x_i = [\mathbf{w}^{(k)}]^T \mathbf{x}$$

The layer function will be:

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} \sigma([\mathbf{w}^{(1)}]^T \mathbf{x}) \\ \sigma([\mathbf{w}^{(2)}]^T \mathbf{x}) \\ \vdots \\ \sigma([\mathbf{w}^{(n)}]^T \mathbf{x}) \end{bmatrix} = \sigma\left(\begin{bmatrix} [\mathbf{w}^{(1)}]^T \mathbf{x} \\ [\mathbf{w}^{(2)}]^T \mathbf{x} \\ \vdots \\ [\mathbf{w}^{(n)}]^T \mathbf{x} \end{bmatrix}\right) = \sigma\left(\begin{bmatrix} (\mathbf{w}^{(1)})^T \\ (\mathbf{w}^{(2)})^T \\ \vdots \\ (\mathbf{w}^{(n)})^T \end{bmatrix} \mathbf{x}\right) = \sigma(\mathbf{W}^T \cdot \mathbf{x})$$

1.3-Feedforward Layered Neural Network

In the artificial neural network discipline, a widely used architecture is the layered network topology. Let L be the number of layers in the network. In general the layers (from layer 2 to layer $L-1$) have the same size between inputs and outputs and the same size between each other so that the weights matrix are $p \times p$ matrices. The first layer captures the input and propagates it throw the network and is represented by a $m \times p$ matrix. The last layer is represented by a $p \times n$ matrix and is responsible for producing the network output. The task of a training algorithm is to determine the entries of the weights



matrices W_1, \dots, W_L associated to each layer according to a given dataset of examples.

2-BACKPROPAGATION ALGORITHM

2.1 - Gradient Descent Training Rule

The gradient descent algorithm uses the information given by the gradient of an error function, to determine the correct unit weights. Suppose we want to train a linear neural unit, mathematically defined by:

$$y = w_0 + \sum_{i=1}^n w_i x_i = \mathbf{w}^T \mathbf{x}$$

where $\mathbf{w} = [w_0 \ w_1 \ , \dots, w_n]^T$, $\mathbf{x} = [1 \ x_1 \ , \dots, x_n]^T$.

Let's now consider a training set having N training examples:

$$D = \left\{ \left(\mathbf{x}^{(1)}, y_d^{(1)} \right), \left(\mathbf{x}^{(2)}, y_d^{(2)} \right), \dots, \left(\mathbf{x}^{(N)}, y_d^{(N)} \right) \right\}$$

where $\mathbf{x}^{(k)}$ is the k -th input example and $y_d^{(k)}$ is the desired output over such input. In the following will be denoted with $y^{(k)}$ the unit output over the input $\mathbf{x}^{(k)}$. A unit is to be considered optimally trained if $y_d^{(k)} - y^{(k)} = 0$ for all $k = 1, \dots, N$.

The error function is commonly defined as the sum of squared error over all training data:

$$E(\mathbf{w}) = \frac{1}{2} \sum_{k=1}^N \left(y_d^{(k)} - y^{(k)} \right)^2$$

The error gradient with respect to unit weights is given by:

$$\nabla E(\mathbf{w}) = \left[\frac{\partial E(\mathbf{w})}{\partial w_0} \quad \frac{\partial E(\mathbf{w})}{\partial w_1} \quad \dots \quad \frac{\partial E(\mathbf{w})}{\partial w_n} \right]^T$$

The generic component of the gradient can be calculated in a few steps:

$$\frac{\partial E(\mathbf{w})}{\partial w_i} = \frac{\partial}{\partial w_i} \left[\frac{1}{2} \sum_{k=1}^N \left(y_d^{(k)} - y^{(k)} \right)^2 \right] = \frac{1}{2} \sum_{k=1}^N \frac{\partial}{\partial w_i} \left(y_d^{(k)} - y^{(k)} \right)^2 = \frac{1}{2} \sum_{k=1}^N 2 \left(y_d^{(k)} - y^{(k)} \right) \frac{\partial}{\partial w_i} (-y^{(k)})$$

where:

$$\frac{\partial}{\partial w_i} (-y^{(k)}) = -\frac{\partial}{\partial w_i} \left(\sum_{i=0}^n w_i x_i^{(k)} \right) = x_i^{(k)}$$

so in definitive:

$$\frac{\partial E(\mathbf{w})}{\partial w_i} = - \sum_{k=1}^N \left(y_d^{(k)} - y^{(k)} \right) x_i^{(k)}$$

The gradient vector indicates the direction of increasing of the error. Because we want to minimize the error we will consider $-\nabla E(\mathbf{w})$. The gradient descent training rule is:

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla E(\mathbf{w})$$

more explicitly:

$$w_i \leftarrow w_i - \eta \frac{\partial E(\mathbf{w})}{\partial w_i}$$

equivalently:

$$w_i \leftarrow w_i + \eta \sum_{k=1}^N (y_d^{(k)} - y^{(k)}) x_i^{(k)}$$

The constant value η is called learning rate and determines the step size to make over the error surface. Big value of η implies fast convergence but with considerable rumor in the found solution, and small η implies slow convergence with good solution.

2.1-Stochastic Gradient Descent & Delta Rule

A modified version of the gradient descent is the *stochastic gradient descent* or also called *incremental gradient descent*. Whereas the classical gradient descent computes the overall error after every weight update, in the stochastic gradient descent will be used a partial error function that measures the error over individual training examples:

$$E^{(k)}(\mathbf{w}) = \frac{1}{2} (y_d^{(k)} - y^{(k)})^2$$

As we can see the previous error function $E(\mathbf{w})$ is the sum of these error functions:

$$E(\mathbf{w}) = \sum_{k=1}^N E^{(k)}(\mathbf{w})$$

The weight update when considering the k-th training example, is:

$$\Delta w_i = \eta (y_d^{(k)} - y^{(k)}) x_i^{(k)}$$

This training rule is also known as *delta rule*.

Let's now consider a neuron with a sigmoid threshold function. The error over the k-th example is:

$$E^{(k)}(\mathbf{w}) = \frac{1}{2} (y_d^{(k)} - y^{(k)})^2 = \frac{1}{2} \left(y_d^{(k)} - \frac{1}{1 + e^{\mathbf{w}^T \cdot \mathbf{x}^{(k)}}} \right)^2$$

The gradient of the error is:

$$\nabla E^{(k)}(\mathbf{w}) = \left[\frac{\partial}{\partial w_1} E^{(k)}(\mathbf{w}) \quad \dots \quad \frac{\partial}{\partial w_n} E^{(k)}(\mathbf{w}) \right]^T$$

Let's calculate the generic component of the error gradient:

$$\begin{aligned}\frac{\partial}{\partial w_j} E^{(k)}(\mathbf{w}) &= \frac{\partial}{\partial w_j} \left(\frac{1}{2} \left(y_d^{(k)} - \sigma(\mathbf{w}^T \mathbf{x}^{(k)}) \right)^2 \right) = - \left(y_d^{(k)} - \sigma(\mathbf{w}^T \mathbf{x}^{(k)}) \right) \cdot \frac{\partial}{\partial w_j} \sigma(\mathbf{w}^T \mathbf{x}^{(k)}) \\ &= - \left(y_d^{(k)} - \sigma(\mathbf{w}^T \mathbf{x}^{(k)}) \right) \cdot \left[\frac{\partial}{\partial (\mathbf{w}^T \mathbf{x}^{(k)})} \sigma(\mathbf{w}^T \mathbf{x}^{(k)}) \right] \cdot \left[\frac{\partial}{\partial w_j} \mathbf{w}^T \mathbf{x}^{(k)} \right]\end{aligned}$$

Recalling the formula of the sigmoid derivative we will have:

$$\frac{\partial}{\partial w_j} E^{(k)}(\mathbf{w}) = - \left(y_d^{(k)} - \sigma(\mathbf{w}^T \mathbf{x}^{(k)}) \right) \cdot \sigma(\mathbf{w}^T \mathbf{x}^{(k)}) (1 - \sigma(\mathbf{w}^T \mathbf{x}^{(k)})) \cdot x_j^{(k)}$$

The update rule is:

$$\Delta w_j = \eta \left(y_d^{(k)} - \sigma^{(k)} \right) \sigma^{(k)} (1 - \sigma^{(k)}) x_j^{(k)}$$

so:

$$w_j \leftarrow w_j + \eta \left(y_d^{(k)} - \sigma^{(k)} \right) \sigma^{(k)} (1 - \sigma^{(k)}) x_j^{(k)}$$

where $\sigma^{(k)} = \sigma(\mathbf{w}^T \mathbf{x}^{(k)})$.

2.2-Backpropagation algorithm

For simplicity we will consider the back-propagation training algorithm in two layer neural network, of m inputs, n outputs and h hidden units (internal units).

First we have to redefine the error function associated to output units of the network. Given a set of training examples:

$$D = \left\{ \left(\mathbf{x}^{(1)}, \mathbf{y}_d^{(1)} \right), \left(\mathbf{x}^{(2)}, \mathbf{y}_d^{(2)} \right), \dots, \left(\mathbf{x}^{(N)}, \mathbf{y}_d^{(N)} \right) \right\}$$

where $\mathbf{x}^{(k)} = [x_1^{(k)} \dots x_m^{(k)}]^T$ is the general input of the network, $\mathbf{y}^{(k)} = [y_1^{(k)} \dots y_n^{(k)}]^T$ is the generic output of the network over $\mathbf{x}^{(k)}$ and $\mathbf{y}_d^{(k)} = [\bar{y}_1^{(k)} \dots \bar{y}_n^{(k)}]^T$ is the required response over $\mathbf{x}^{(k)}$, the error is defined the sum of all the squared errors associated to each output unit:

$$E(\mathbf{w}) = \frac{1}{2} \sum_{k=1}^N \sum_{j=1}^n \left(\bar{y}_j^{(k)} - y_j^{(k)} \right)^2$$

As in the case of training single units, the gradient descent can be used for minimizing the error by using its gradient. The difference is that in single unit models, the error's gradient has a single minimum. In the case of multilayer networks the error may present multiple local minima and may be difficult reach the global minima.

The Back-Propagation algorithm described in [4] is articulated in the following steps:

- create a feed-forward network with m input units, h hidden units and n output units
- initialize all network weights to some small random value²

² In the character recognition application, the weights were initialized by a random uniform distribution in the interval [0.5;0.5]. A certain grade of randomness was necessary. In the case of all zero weights there was practically no change of the error function during the training.

- Until termination condition is met, do
 - for each $(\mathbf{x}, \mathbf{y}_d) \in D$, do
 1. Propagate the input \mathbf{x} throw the network and compute the output y_u of every unit u .
 2. For each output unit k , calculate its error term: $\delta_k = y_k(1 - y_k)(\bar{y}_k - y_k)$
 3. For each hidden unit h , calculate its error term: $\delta_h = y_h(1 - y_h) \sum_{k \in \text{out}} w_{kh} \delta_k$
 4. Update each network weight: $w_{ji}^{(t+1)} = w_{ji}^{(t)} + \Delta w_{ji}^{(t)} + \mu_{ji}^{(t)}$, where $\Delta w_{ji}(t) = \eta \delta_j(t) x_{ji}$ and $\mu^{(t-1)} = \alpha \Delta w_{ji}^{(t-1)}$.

The Back-Propagation algorithm is used in our application to train a neural net for alphanumeric recognition. The algorithm is implemented in MatLab and you can find it in <..\Alphanumeric Recognition\2-Back Propagation\backPropagationAlgorithm.m>. The code is a general implementation of the algorithm, so it can be used for every kind of recognition problem. The only restriction is that the code is prepared for training exactly two layered network, with unlimited number of units. The presence of only two layers let us make updates of weights and biases separately. Weights parameters are generated randomly in the interval $[-0.5; +0.5]$. The biases are initialized all to zero. The directory <..\Alphanumeric Recognition\1-Neural Net Functions> contains some accessory scripts of homonym functions:

- [initUnit.m>initUnit\(...\)](#) : initializes a neuron unit weights vector
- [neuron.m>neuron\(...\)](#) : returns the output of a single neuron given an input vector, a weight vector and the type of threshold function
- [initLayer.m>initLayer\(...\)](#) : initializes a layer matrix weights
- [layer.m>layer\(...\)](#) : returns the output vector of a layer, given the input vector, the weights matrix and the threshold functions for the layer units
- [initNetwork.m>initNetwork\(...\)](#) : initialize the weights matrix and bias vectors of a two layer network
- [network2.m>network2\(...\)](#) : returns the output vector of a two layered network given the input vector, the weights matrices, the biases vectors and the typology of threshold function.

In the figures on the next two pages are presented some different error evolution, during the back-propagation training for the character recognition. It was used a logarithmic scale because in general, the error first has a great magnitude that falls down to fast. The different training presented will be explained in details in the next session. Here we want to get a look of the training procedure in general.

Another crucial point is the termination condition. As explained in the literature, earlier termination may cause non accurate discrimination of the data and later termination may cause over-fitting of patterns that implies loss of generalization power of the network. Theoretically, a good network is the network trained until the error becomes zero and the trained patterns constitute the total class of objects that may be presented. This is not the approach used in machine learning discipline in general and neural networks in particular. Another possibility is to find a mathematical model that describes all patterns. Having this mathematical generalization³ has the consequence to not need any more learning algorithms.

³ Modeling the nature by a mathematical model is an unrealistic idea. Let's suppose we have this mathematical model of the nature. Because this model is itself part of the nature, consequently is mathematically described inside the model. It was proven that mathematical model that describes themselves are paradoxal (exm. Russell's Paradox) or non-decidable(exm. termination of the Turing Machine). These facts make Machine Learning necessary for understanding the nature using the power of the modern technologies. Also this facts makes the nature pretty; there are non-demonstrable verities.

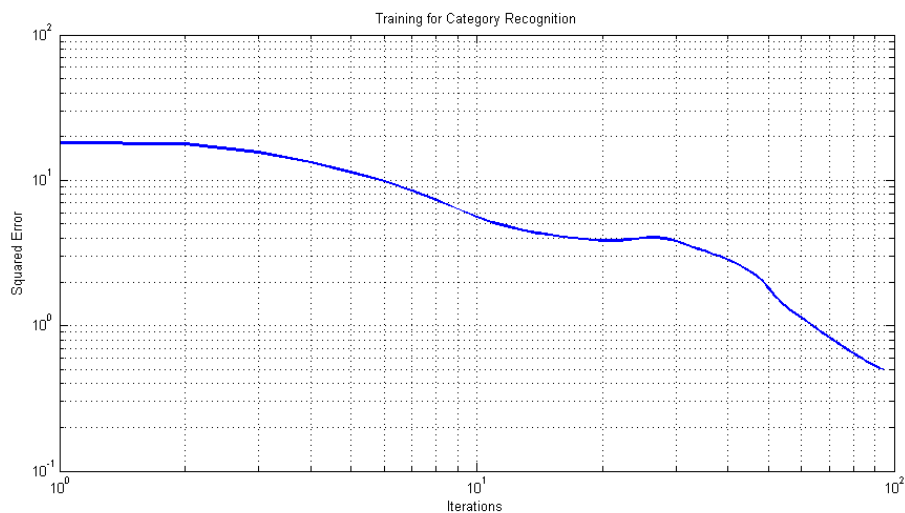


Figure 3- results obtained with: in size=400, out size=3 and 248 training examples

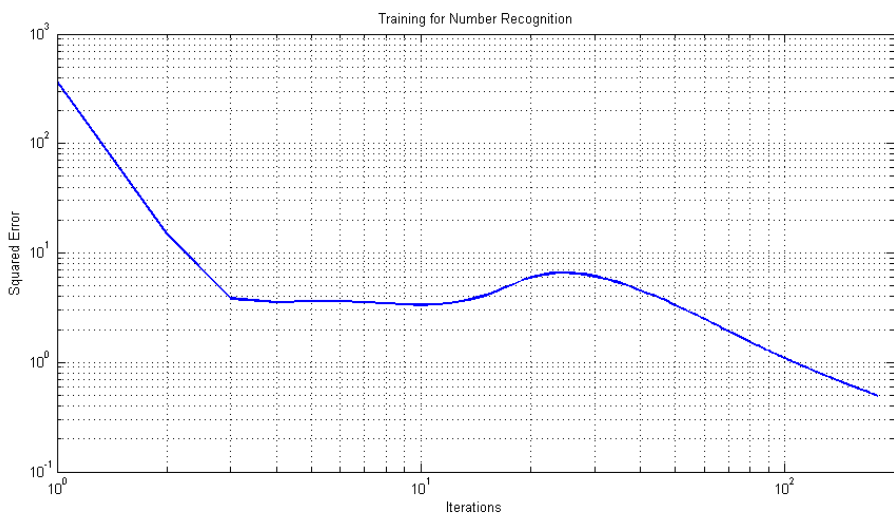


Figure 4- results obtained with: in size=400, out size=10 and 248 training examples

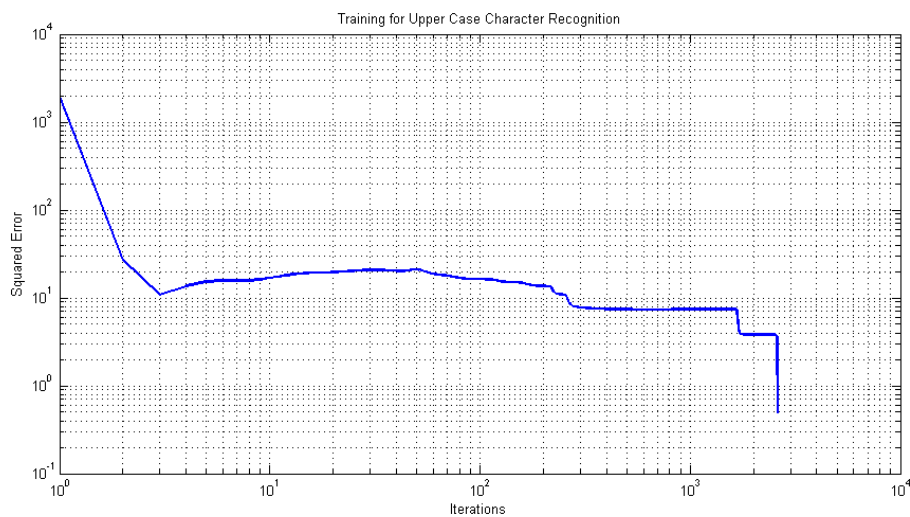


Figure 5- results obtained with: in size=400, out size=26 and 248 training examples

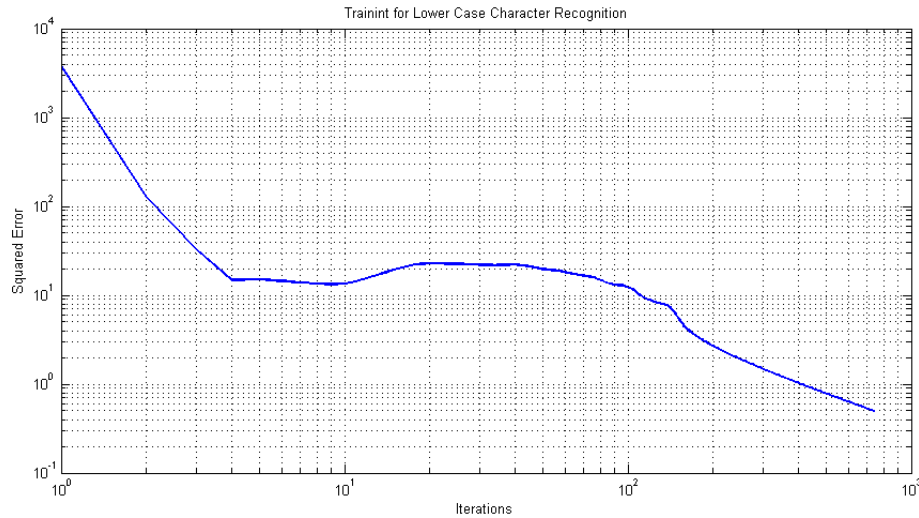


Figure 6- results obtained with: in size=400, out size=26 and 248 training examples

3-ALPHANUMERIC RECOGNITION

We are interested on the character recognition problem, in particular the *alphanumeric typescript recognition*. The system implemented is a small experimental application. Why we call it experimental? First, all the mechanism is implemented in MatLab. Second, the recognition of alphanumeric is satisfactory but not good.

Several applications are developed for this purpose, around the World. One typology of these applications recognizes only type-script characters; others can recognize⁴ even manuscript characters. The manuscript recognizing problem is still open. In the last section of this paper will be presented a neural model for manuscript recognition, called Neocognitron.

3.1-Typescript Recognition

In this section is presented the approach used for the typescript recognition. The goal is to recognize single characters presented in digital image format. The recognition of the image text is done by decomposing it in single character images.

The recognition model follows the neural network model. The char image constitutes the input of the network, which gives a response about what have recognized. There are used four neural networks organized in two levels. The first level there is only one network, responsible of determine the typology⁵ of the presented character. The second level includes three networks, able to recognize characters from only one typology.

⁴ Manuscript recognizers are present in modern touch screen cell phones. Anyone can test the difficulties of recognition. One reason may be the economic cost of the software, however in general alphanumeric applications are already infantile.

⁵ Characters typologies are simply three, (1) Arab numbers, (2) lower case Latin characters and (3) upper case Latin characters. This class discrimination was chosen for simulating the human concept discrimination the number concept and the letter concept. However the lower case and uppercase concepts are not so far for humans, but in our applications this distinction has been done for increasing the shape discrimination power.

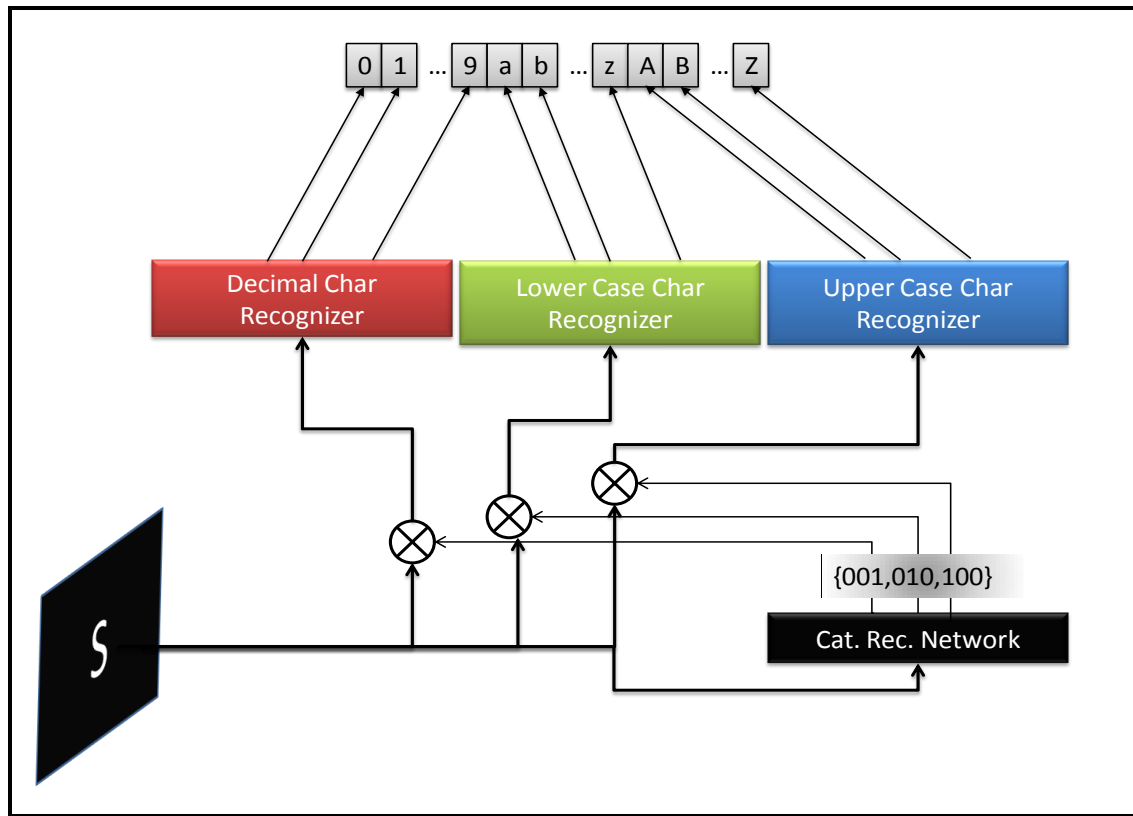


Figure 7-Structure of the typescript recognizer network.

In Table 1-Networks Parameters are represented the networks parameters that necessary⁶ (for sufficiently good recognition), such as number of hidden units, size of the input image, number of training examples used, and number of necessary iterations for a good performance.

Table 1-Networks Parameters

Network ⁷	Input Size		Number of Hidden Units		Output Size	Iterations until convergence ⁸ ($\eta = 0.05$)	
	Min	Max	Min	Max		$\alpha = 0.8$	$\alpha = 0$
CRN	10x10	30x30	10	35	3	50-300	300-1500
NRN	--/--	--/--	20	50	10	200-500	500-2000
LRN	--/--	--/--	30	80	26	400-1500	600-2100
URN	--/--	--/--	30	80	26	400-1500	600-2100

⁶ The entries of the table are extracted empirically during test. There was not a specific examination in this sense.

⁷ CRN = Category Recognition Network

NRN = Decimal Recognition Network

LRN = Lower Cases Recognition Network

URN = Upper Cases Recognition Network

⁸ The convergence criterion is reached when the sum of squared error becomes smaller than 0.5. A maximum number of iterations was forced to 10'000. Even though this maximum was not reached until the dataset was obtained by the most common typescript fonts such as "Verdana", "Times New Roman" etc. The error converges slowly when in the training data are inserted fonts similar to manuscript characteristic, this because of the great difference of data of the same concept.

The output size was chosen to be of the same size of the concept set cardinality, in particular for each network we have:

- CRN: 3 binary outputs {001,010,100} respectively associated to decimal, lower case chars and upper case chars.
- NRN: 10 output $\left\{ \left(\begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}_{10} , "0" \right) , \left(\begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \\ 0 \end{bmatrix}_{10} , "1" \right) , \dots , \left(\begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \\ 0 \end{bmatrix}_{10} , "8" \right) , \left(\begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}_{10} , "9" \right) \right\}$
- LRN: 26 $\left\{ \left(\begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}_{26} , "a" \right) , \left(\begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \\ 0 \end{bmatrix}_{26} , "b" \right) , \dots , \left(\begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \\ 0 \end{bmatrix}_{26} , "y" \right) , \left(\begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}_{26} , "z" \right) \right\}$
- URN: 26 $\left\{ \left(\begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}_{26} , "A" \right) , \left(\begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \\ 0 \end{bmatrix}_{26} , "B" \right) , \dots , \left(\begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \\ 0 \end{bmatrix}_{26} , "Y" \right) , \left(\begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}_{26} , "Z" \right) \right\}$

Regarding the input size, it was chosen to be flexible, accepting an arbitrary image dimension. This does not mean that the network will work properly with any image size.

The resizing was not a simple scale operation. First the character is centered in a squared image with size equal to the length of the largest side of the original rectangle. Next the image is resized. This allows you to retain the original proportions of the character.

During tests was used an input size between 20X20. Images matrices were converted in linear arrays. Training examples was extracted from four typescript fonts (Figure 9).



Table 2 presents some simple tests of recognition. For more details see the [test results](#) generated with MatLab. The first table presents an image from a webcam of "La Sapienza University" logo. The image passed through some rectification steps⁹ necessary for extracting character images, recognizable for the network. After, the character images were passed to the network for recognition.

0	1	2	3	4	5	6	7	8	9	0
0	1	2	3	4	5	6	7	8	9	0
0	1	2	3	4	5	6	7	8	9	0
0	1	2	3	4	5	6	7	8	9	0
a	b	c	d	e	f	g	h	i	j	k
a	b	c	d	e	f	g	h	i	j	k
a	b	c	d	e	f	g	h	i	j	k
a	b	c	d	e	f	g	h	i	j	k
A	B	C	D	E	F	G	H	I	J	K
A	B	C	D	E	F	G	H	I	J	K
A	B	C	D	E	F	G	H	I	J	K
A	B	C	D	E	F	G	H	I	J	K

Figure 8 - A possible dataset for training

⁹ The image was first filtered to remove noise, then was used a clustering step over the color to remove the background and finally the remaining image was converted in binary image.

Table 2-Recognition tests

Text Image	Recognition Result	Recognizing with LRN ¹⁰	Recognizing with URN	Recognition with NRN
original  converted for recognition 	8 A P I E N Z A	*****	S A P I E N Z A	*****
Machine Learning 2010	MaC h i n e Lea L n i n g 20 I 0	* a c h i n e * e a r n i n g ****	M ***** L ***** ****	***** ***** 2010
ARTIFICIAL NEURAL NETWORK	A R T I F I C I A L N E U R A L N E T w o R K	****	ARTIFICIAL NEURAL NETWORK	****
1601201031122009	1 6 0 1 2 0 1 0 3 1 1 2 2 0 0 s	****	1 6 0 1 2 0 1 0 3 1 1 2 2 0 0 9	****

The letter "S" was recognized as "8". We can easily understand that the problem stays on the CRN network, because it was recognized a decimal character instead of an upper case char. To be more convinced let se the result given from URN. There was recognized all the characters of the image. The same problem emerges in the second test where we have mixed the three kinds of characters: (1) the letter "c" was recognized as "C" but this is not a problem because the two letters are similar¹¹ (2) the letter "r" was recognized as "L" and this is a problem because this two letters have different shapes¹² (3) the decimal "1" was recognized as "l", the same phenomenon as in (1). The same problems are presented in the rest of the tests.

The ambiguity between the different characters with similar shapes is not just a problem of artificial recognition systems. It also exists in human recognition. However, the human brain is not limited to recognize the characters one by one. It is true that the characters are presented as individual images, but in later stages of recognition is analyzed the context of the words, phrases or discussions. It is possible to increase the power of the recognizer, connecting it to a database of correct words. The procedure can be divided into the following steps:

1. Capture the image from the retina¹³.
2. Determine the textual region on the image.
3. Decompose the text image to words image¹⁴.

¹⁰ If we get the response by using a particular network from NRN, LRN and URN we will have false results for characters non recognizable by each net. For this purpose the outer-class characters during a particular type of recognition are replaced with "*".

¹¹ Let anyone try to recognize an isolated character such as "c" or "w", and determine if it is in lower case or upper case.

¹² Maybe the network was not trained enough. These situations may be used for used for a second training phase of the network.

¹³ electronic camera

4. For each word image try get single character images and recognize the associated character concept concatenating in order characters from the same word image.
5. For each reconstructed word try to associate it with a concept word by searching the nearest word stored in the database¹⁵.
6. Return the constructed text.

It may be possible apply another stage of the recognition by applying another model for checking the correctness of reconstructed phrases. Making use of grammatical property of the particular language used.

Obviously, this system can be used even for recognizing manuscript text. Moreover, this approach will increase the recognition power of manuscript texts by increasing the interval of allowed deformation for written symbolic characters.

In definitive, let's take another example of computation power of the human brain, by getting a look of this phrase:

acrodcing to smoe sicetnists, for huamn banes, gevin a wrod , it is not ipmotrant the odrer of how
leterts are presented but simply the prenesce

3.2-Manuscript Recognition

Let now return to the initial problem, recognize single symbols from images. In this section will be presented a model of manuscript characters recognizer, known as *neocognitron*. This section is a synthesis of [3]

The neocognitron is a hierarchical network consisting of several layers of neuron-like cells. The cells are organized in layers feed-forward connected. Some of these connections are variable, and can be modified by learning. The symbols¹⁶ are initially decomposed by extracting some features such as angles, lines etc. These features are propagated through the network and analyzed with the same methodology, by extracting features. In Figure 10 is shown the structure of the neocognitron.

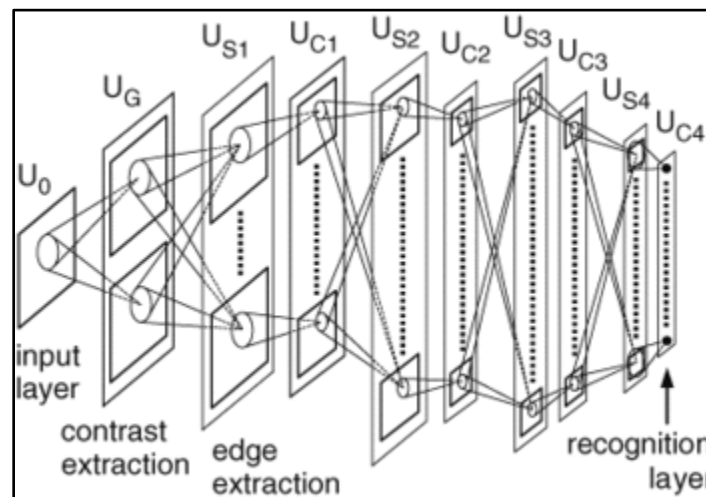


Figure 9 - Hierarchical network structure of the neocognitron.

¹⁴ Decomposition may be done by using a recognizer for delimiter symbols such as ",", "." etc. Either blank spaces between words image may be used.

¹⁵ The query to the database may pass through neural network that models the feeling of familiarity. In case of query for words that's look familiar up to a certain level will be returned a positive answer without making use of the DB.

¹⁶ Neocognitron is able to recognize even non alphanumerical symbols, such as japans characters, Hindi characters, simple geometric shapes etc.

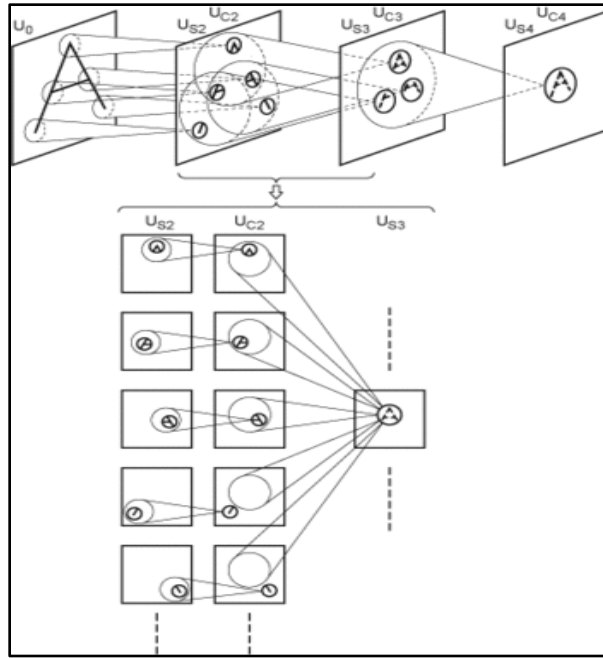


Figure 10 - The process of pattern recognition in the neocognitron.

The input layer is a two dimensional array of cells corresponding to the retina. Each succeeding stage has two layers consisting of cells called S and cell called C.¹⁷ S cells are responsible for feature extraction and C cells are responsible for allowing positional errors. The layer of C cells at the highest stage is the recognition layer, representing the final result. The notations U_{S_i} and U_{C_i} are used to indicate the layers of S cells and C cells in the i -th stage, respectively. The input layer is denoted by U_0 .

Each layer of S cells or C cells is divided into subgroups, called cell planes, according to the feature they are sensible. All cells in one cell plane share the same spatial input connections. Every cell detects the same feature but from a slightly different position. Every cell receives its input through a *receptive field*. For example, S cells does not search for a feature in the entire input plane. Instead it is assigned a small area of the input plane. and the S cell fires only if the corresponding feature is available. The same concept is applied to C cells.

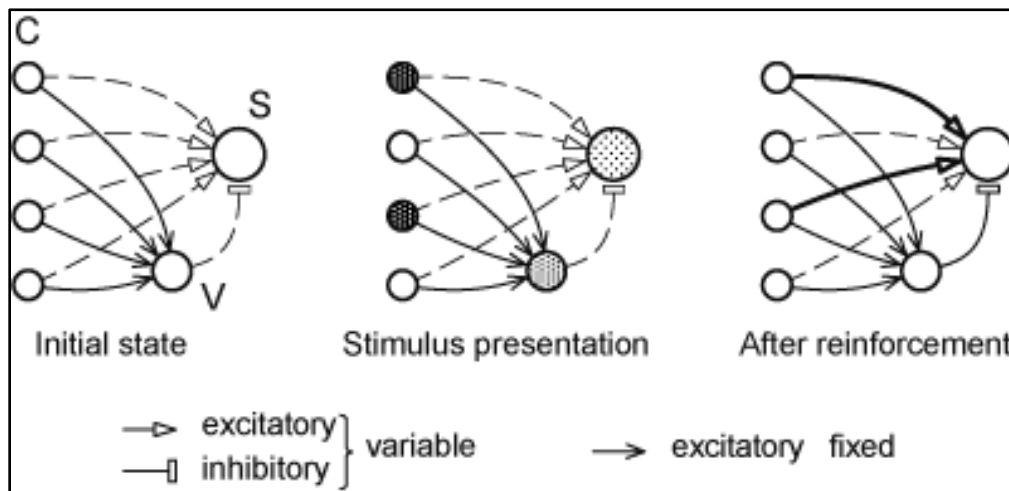


Figure 11 - Connections between cells in the network.

¹⁷ "S" stays for simple and "C" stays for complex.

Figure 12 shows the connections between C cells and S cells. A layer of subsidiary V cells are inserted. The input of every V cell is the same as the input of corresponding S cell. A particular input is inserted to C cells that makes the role of inhibitory connection. Suppose a particular S cell receives excitatory input from a group of C cells whose output are u_1, \dots, u_n , and an inhibitory input from a V cell whose output is v , then the output of the S cell is defined by:

$$u = r\phi\left(\frac{1 + \sum_{i=1}^n \alpha_i u_i}{1 + \frac{1}{1+r}bv} - 1\right)$$

where r is a positive constant that determines the efficiency of the inhibition by the V cell, α_i 's are the weights of the interconnections and b is the weight applied to the inhibitory connection. The function ϕ is defined as:

$$\phi(t) = \begin{cases} t, & t \geq 0 \\ 0, & \text{otherwise} \end{cases}$$

In Figure 13 is shown an example of response of the neocognitron with two stages. The input pattern is recognized correctly as '5'. We can see what that in level U_{C1} are projected particular feature for each C cell plane. The second stage repeats the same process over the output of the previous layer. The figure shows only the C cell planes, that have the property to reassemble the features extracted from S cell layer.

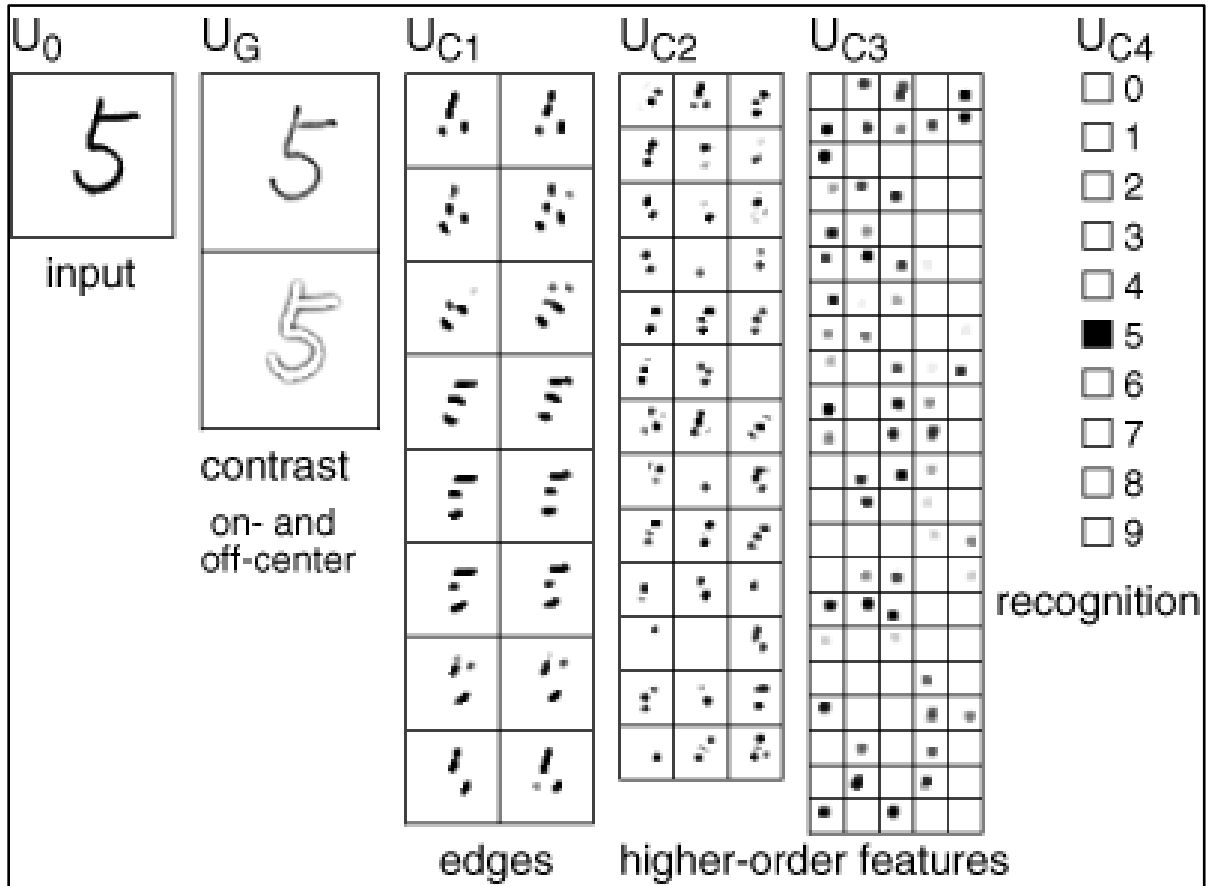


Figure 12 - An example of the response of a neocognitron that has been trained to recognize handwritten digits.

References

1. **Biological object recognition** [Online] / auth. Kreiman Gabriel // Scholarpedia. - 2008. - http://www.scholarpedia.org/article/Biological_object_recognition.
2. **Handwriting Recognition** [Online] // Wikipedia. - http://en.wikipedia.org/wiki/Handwriting_recognition.
3. **Hanwritten Alphanumeric Character Recognition by the Neocognitron** [Journal] / auth. Fukushima Kunihiro and Wake Nobaki. - [s.l.] : IEEE Transactions On Neural Networks, 1991. - 3 May : Vol. 2. - pp. 355-365.
4. **Machine Learning** [Book] / auth. Mitchel Tom. - [s.l.] : McGraw-Hill, 1997. - 1997.
5. **Neocognitron** [Online] / auth. Fukushima Kunihiro // Scholarpedia. - 2007. - <http://www.scholarpedia.org/article/Neocognitron>.
6. **Neural Networks - A Systematic Introduction** [Book] / auth. Rojas Raul.
7. **Pattern Recognition & Machine Learning** [Book] / auth. Bishop Cristopher M. - [s.l.] : Springer, 2006.