

6 Selected Applications of the Multi-layer Perceptrons

6.1 Image Coding using Multi-layer Perceptrons

In this example we study an application of a two-layer feed-forward neural network and the back-propagation algorithm in image coding. In particular, we will aim at a coding scheme which delivers some degree of image compression.

A general concept of image coding using a feed-forward neural network consists of the following points:

- An image, F , is divided into $r \times c$ blocks of pixels. Each block is then scanned to form an input vector $\mathbf{x}(n)$ of size $p = r \times c$.

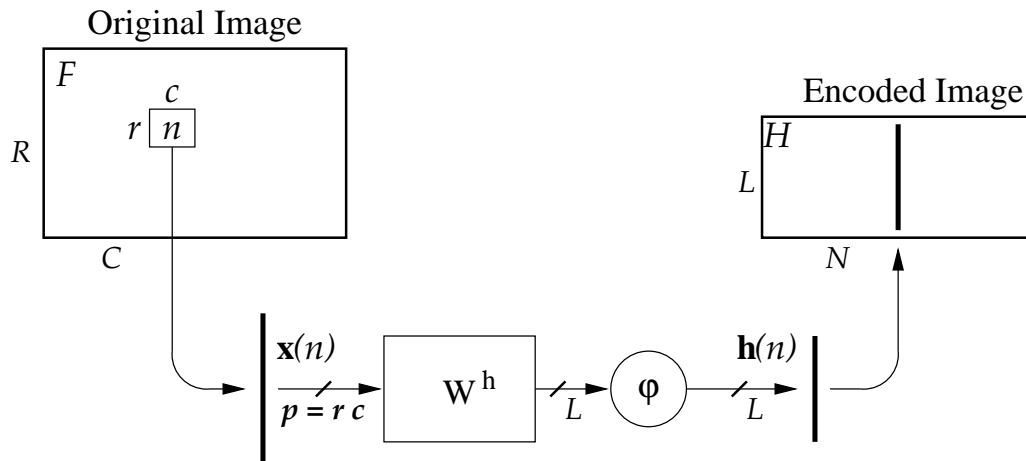


Figure 6–1: A perceptron layer in image encoding

- Assume that the hidden layer of the neural network consists of L neurons each with p synapses, and that it is characterised by the appropriately selected weight matrix W^h .

Pass all N blocks of the original image through the hidden layer to obtain the hidden signals, $\mathbf{h}(n)$, which represent encoded input image blocks, $\mathbf{x}(n)$.

If $L < p$ such coding delivers image compression.

- Assume that the output layer consists of $m = p = r \times c$ neurons, each with L synapses. Let W^y be an appropriately selected output weight matrix.

Pass all N hidden vectors, $\mathbf{h}(n)$, representing an encoded image, H , through the output layer to obtain the output signal, $\mathbf{y}(n)$. Re-assemble the output signals into $p = r \times c$ image blocks to obtain a re-constructed image, F^r .

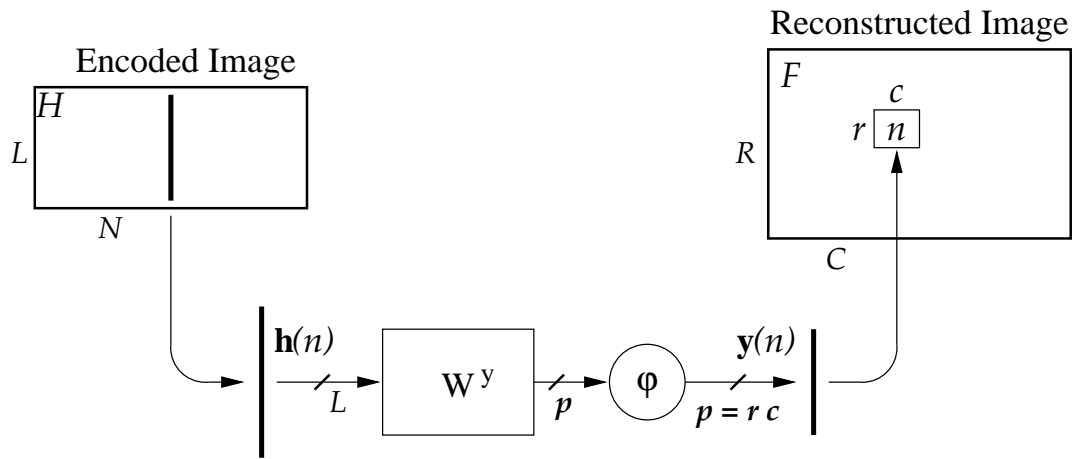


Figure 6–2: Image decoding by a perceptron

The quality of image coding is typically assessed by the Signal-to-Noise Ratio (SNR) defined as

$$\text{SNR} = 10 \log \frac{\sum_{i,j} (F_{i,j})^2}{\sum_{i,j} (F_{i,j}^r - F_{i,j})^2}$$

- Training is conducted for a representative class of images using the back-propagation algorithm.
- Once the weight matrices have been appropriately selected, any image can be quickly encoded using the W^h matrix, and then decoded (reconstructed) using the W^y matrix.

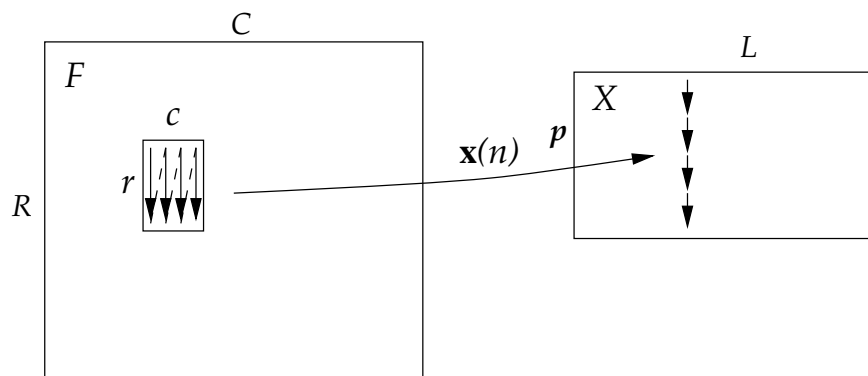
6.1.1 Training procedure

The simplest image coding system consists of a two-layer perceptron. During training procedure data from a representative image, or a class of images is encoded into a structure of the hidden and output weight matrices.

Assume that an image, F , used in training is of size $R \times C$ and consists of $r \times c$ blocks.

1. The first step is to convert a block matrix F into a matrix X of size $p \times N$ containing training vectors, $\mathbf{x}(n)$, formed from image blocks. Note that

$$p = r \cdot c, \text{ and } p \cdot N = R \cdot C$$



Use the following MATLAB function to perform this conversion:

```
X = blkM2vc(F, [r c]);
```

```
function vc = blkM2vc(M, blkS)
[rr cc] = size(M) ;
r = blkS(1) ; c = blkS(2) ;
if (rem(rr, r) ~= 0) | (rem(cc, c) ~= 0)
    error('blocks do not fit into matrix')
end
nr = rr/r ; nc = cc/c ; rc = r*c ;
```

```

vc = zeros(rc, nr*nc);
for ii = 0:nr-1
    vc(:,(1:nc)+ii*nc) = reshape(M((1:r)+ii*r,:),rc,nc);
end

```

2. As a target data use the input data, that is:

$$D = X$$

3. Train the network until the mean squared error, J , is sufficiently small. The matrices W^h and W^y will be subsequently used in the image encoding and decoding steps.

6.1.2 Image encoding

As explained in Figure 6–1 the hidden-half of the two-layer perceptron is used to encode images. The encoding procedure can be described as follows:

$$F \longrightarrow X, \quad H = \varphi(W^h \cdot X)$$

where X is an encoded image F .

6.1.3 Image decoding

With reference to Figure 6–2 the image is decoded (reconstructed) using the output-half the two-layer perceptron. The decoding procedure can be described as follows:

$$Y = \varphi(W^y \cdot H), \quad Y \longrightarrow \hat{F}$$

The conversion of the vectors of the reconstructed image stored in the $p \times N$ matrix Y into blocks of the reconstructed image can be performed using the following MATLAB function:

```
Fr = vc2blkM(Y, r, R);
```

where

```
function M = vc2blkM(vc, r, rM)
%vc2blkM Reshaping a matrix vc of rc by 1 vectors into a
%          block-matrix M of rM by cM size
% Each rc-element column of vc is converted into a r by c
% block of a matrix M and placed as a block-row element
[rc nb] = size(vc) ; pxls = rc*nb ;
if ( (rem(pxls, rM) ~= 0) | (rem(rM, r) ~= 0) )
    error('incorrect number of rows of the matrix')
end
cM = pxls/rM ;
if ( (rem(rc, r) ~= 0) | (rem(nb*r, rM) ~= 0) )
    error('incorrect block size')
end
c = rc/r ;
xM = zeros(r, nb*c);
xM(:) = vc ;
nrb = rM/r ;
M = zeros(rM, cM);
for ii = 0:nrb-1
    M((1:r)+ii*r, :) = xM(:, (1:cM)+ii*cM) ;
end
```

6.2 Paint-Quality Inspection

Adapted from (Freeman and Skapura, 1991)

Visual inspection of painted surfaces, such as automobile body panels, is currently a very time-consuming and labor-intensive process. To reduce the amount of time required to perform this inspection, one of the major U.S. automobile manufacturers reflects a laser beam off the painted panel and on to a projection screen. Since the light source is a coherent beam, the amount of scatter observed in the reflected image of the laser provides an indication of the quality of the paint finish on the car.

A poor paint job is one that contains ripples, looks like “orange peel”, or lacks shine. A laser beam reflected off a panel with a poor finish will therefore be relatively diffuse.

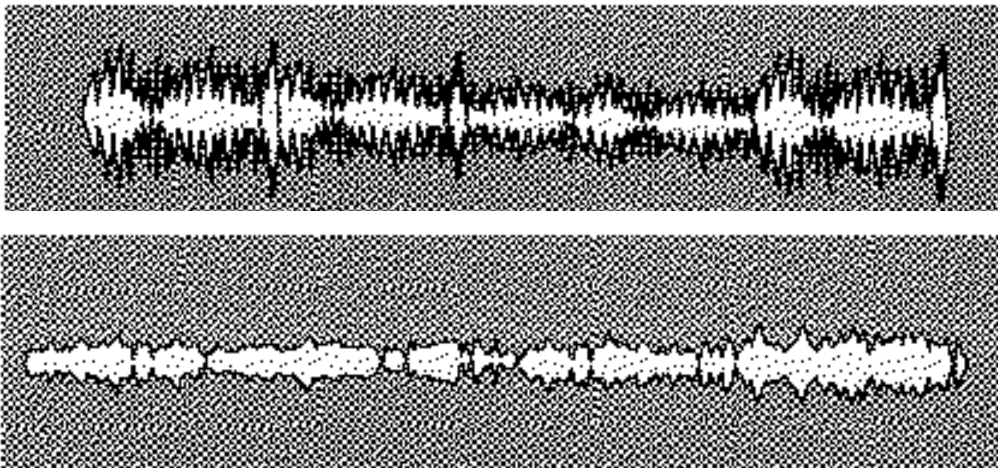


Figure 6–3: The scatter typically observed when a laser beam is reflected off painted sheet-metal surfaces: (top — a poor-quality paint finish, bottom — a better-quality paint finish).

Conversely, a good-quality paint finish will be relatively smooth and will exhibit a bright luster. A laser light reflected off a high-quality paint finish will appear to an observer as very close to uniform throughout its image.

A neural network, a two-layer perceptron in this case, is used to capture the expertise of the human inspectors scoring the paint quality from observation of the reflected laser images.

The block-diagram of the Automatic Paint QA System is presented in Figure 6–4

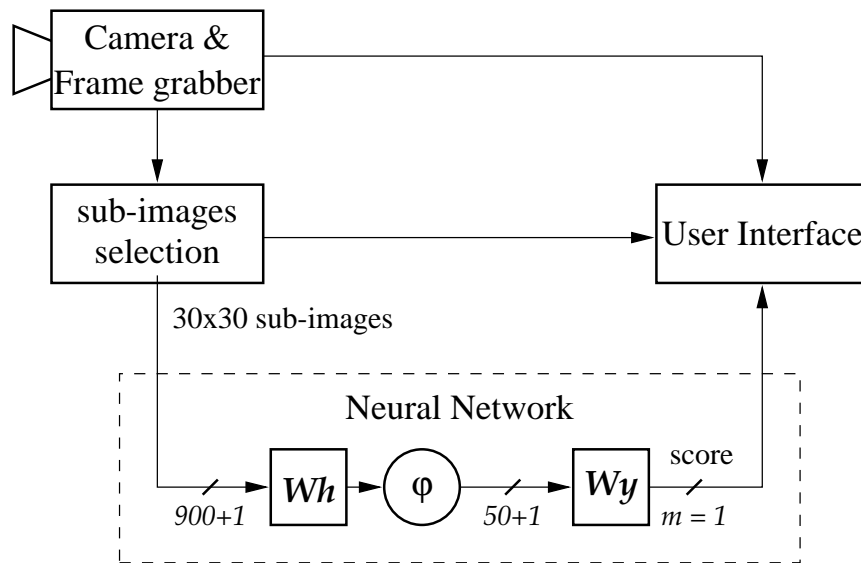


Figure 6–4: The block-diagram of the Automatic Paint QA System

Images of the reflected laser beam are recorded by a camera and an associated frame grabber. Each image contains 400-by-75 8-bit pixels.

To keep the size of the network needed to solve the problem manageable, we elected to take 10 sub-images from the snapshot, each sub-image consisting of a 30-by-30-pixel square centered on a region of the image with the brightest intensity. These 8-bit pixels are input to the neural network. In addition there is one biasing input, therefore, $p = 901$.

The hidden layer consists of $L = 50$ neurons, hence the hidden-matrix, W^h is 900×50 , and there are $900 \times 50 = 45000$ synapses in the hidden layer. A unipolar sigmoidal function is used.

A single output signal from the network represents a numerical score in the range of 1 through 20 (a 1 represented the best possible paint finish; a 20 represented the worst).

The output layer is **linear** and the output matrix W^y has a size 51×1 (there is a biasing input to the output layer).

Once the network was constructed (and trained), 10 sub-images were taken from the snapshot using two different sampling techniques.

In the first test, the samples were selected randomly from the image (in the sense that their position on the beam image was random).

In the second test, 10 sequential samples were taken, so as to ensure that the entire beam was examined.

In both cases, the input sample was propagated through the trained BPN, and the score produced as output by the network was averaged across the 10 trials. The average score, as well as the range of scores produced, were then provided to the user for comparison and interpretation.

Training the Paint QA Network

At the time of the development of this application, (1988) this network was significantly larger than any other network we had yet trained.

Consider the size of the network used: 901 inputs, 51 hidden, 1 output, producing a network with 45 101 synapses (connections), each modeled as a floating-point number.

Similarly, the unit output values were modeled as floating-point numbers, since each element in the input vector represented a pixel intensity value (scaled between 0 and 1), and the network output unit was linear.

The number of training patterns with which we had to work was a function of the number of control paint panels to which we had access (18), as well as of the number of sample images we needed from each panel to acquire a relatively complete training set (approximately 6600 images per panel).

During training, the samples were presented to the network randomly to ensure that no single paint panel dominated the training.

From these numbers, we can see that there was a great deal of computer time consumed during the training process.

For example, one training epoch (a single training pass through all training patterns) required the host computer to perform approximately 13.5 million connection updates, which translates into roughly 360,000 floating-point operations (FLOPS) per pattern (2 FLOPS per connection during forward propagation, 6 FLOPS during error propagation), or 108 million FLOPS per epoch.

You can now understand why we have emphasized efficiency in our simulator design.

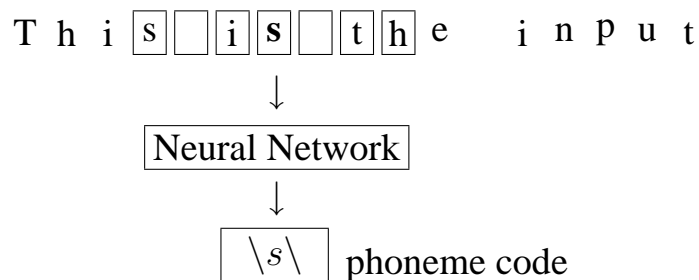
We performed the network training for this application on a dedicated LISP computer workstation. It required almost 2 weeks of uninterrupted computation for the network to converge on that machine.

However, once the network was trained, we ported the paint QA application to an 80386-based desktop computer by simply transferring the network connection weights to a disk file and copying the file onto the disk on the desktop machine. Then, for demonstration and later paint QA applications, the network was utilized in a production mode only.

6.3 NETtalk

Sejnowski and Rosenberg, 1987)

The NETtalk project aimed at training a network to pronounce English text. The conceptual structure of the network is as follows:



A character from a text and its three preceding and three following characters are entered into a neural network which generates a phoneme code for the central character.

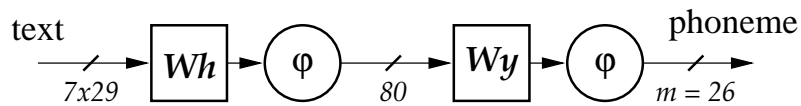
The phoneme code can be sent to a speech generator giving the pronunciation of the central letter from the input window.

Network structure

There are 29 English letters (including punctuation) and each letter is coded in a 1-of-29 code. Therefore, there are $p = 7 \times 29 = 203$ binary inputs to the network.

Similarly, there are 26 different phonemes, hence, the network has 26 binary outputs.

In addition, 80 hidden neurons are employed.



Training

During training, the desired data were supplied by a commercially available DEC-talk, which is based on hand-coded linguistic rules.

The network was trained on 1024 words, obtaining intelligible speech after 10 training epochs and 95% accuracy after 50 epochs.

References

- [DB98] H. Demuth and M. Beale. *Neural Network TOOLBOX User's Guide. For use with MATLAB*. The MathWorks Inc., 1998.
- [FS91] J.A. Freeman and D.M. Skapura. *Neural Networks. Algorithms, Applications, and Programming Techniques*. Addison-Wesley, 1991. ISBN 0-201-51376-5.
- [Has95] Mohamad H. Hassoun. *Fundamentals of Artificial Neural Networks*. The MIT Press, 1995. ISBN 0-262-08239-X.
- [Hay99] Simon Haykin. *Neural Networks – a Comprehensive Foundation*. Prentice Hall, New Jersey, 2nd edition, 1999. ISBN 0-13-273350-1.
- [HDB96] Martin T. Hagan, H Demuth, and M. Beale. *Neural Network Design*. PWS Publishing, 1996.
- [HKP91] Hertz, Krogh, and Palmer. *Introduction to the Theory of Neural Computation*. Addison-Wesley, 1991. ISBN 0-201-51560-1.
- [Kos92] Bart Kosko. *Neural Networks for Signal Processing*. Prentice-Hall, 1992. ISBN 0-13-617390-X.
- [Sar98] W.S. Sarle, editor. *Neural Network FAQ*. Newsgroup: comp.ai.neural-nets, 1998. URL: <ftp://ftp.sas.com/pub/neural/FAQ.html>.