Learning Optimal Linear Filters for Edge Detection

by

Xiaofang Sun

B.Sc., Peking University, China, 1989

A Thesis Submitted in Partial Fulfillment of
the Requirements for the Degree of
Master of Science

in

The Faculty of Graduate Studies
Department of Computer Science

We accept this thesis as conforming
to the required standard

University of British Columbia
August 1991
©Xiaofang Sun, 1991
In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of Computer Science

The University of British Columbia
Vancouver, Canada

Date July 30, 1991
Abstract

Edge detection is important both for its practical applications to computer vision as well as its relationship to early processing in the visual cortex. We describe experiments in which the back-propagation learning algorithm was used to learn sets of linear filters for the task of determining the orientation and location of edges to sub-pixel accuracy. A model of edge formation was used to generate novel input-output pairs for each iteration of the training process. The desired output included determining the interpolated location and orientation of the edge. The linear filters that result from this optimization process bear a close resemblance to oriented Gabor or derivative-of-Gaussian filters that have been found in primary visual cortex. In addition, the edge detection results appear to be superior to the existing standard edge detectors and may prove to be of considerable practical value in computer vision.
Contents

Abstract

Contents

List of Figures

Acknowledgement

1 Introduction 1

2 Background and Related Work 5
  2.1 Previous Research on Edge Detection 5
      2.1.1 Marr and Hildreth's Edge Detector 6
      2.1.2 Canny's Edge Detector 8
      2.1.3 Problem with Multiple Scales 10
  2.2 Neural Network 11
      2.2.1 The Basic Neuron Model 12
      2.2.2 The Neural Network Models 14
      2.2.3 Back-propagation and its Applications 14
      2.2.4 Plaut and Hinton's Paper 16

3 A Learning Approach for Edge Detection 18
  3.1 1-D Edge Detection 18
  3.2 The Neural Network Architecture to Perform 2-D Edge Detection 23
  3.3 Input-Output Training Data Generation 24
      3.3.1 Input Data Generation 24
      3.3.2 Desired Output Design 32
  3.4 Training Algorithm and Training Process 34
3.4.1 Back-propagation Algorithm .......................... 34
3.4.2 Rescaling Some Variables ............................ 38
3.4.3 Training Process ......................................... 38
3.5 Thinning Algorithm and Subpixel Calculation .......... 40

4 Results and Hidden Units Analysis .......................... 46
  4.1 Experimental results ...................................... 46
  4.2 Hidden Units Analysis ..................................... 55
      4.2.1 Biological Visual Primitives ....................... 55
      4.2.2 Neural Network Weighting Patterns ................. 59

5 Discussion ....................................................... 65
  5.1 Summary ................................................... 65
  5.2 Problems .................................................. 66
  5.3 Future Improvements ...................................... 67
List of Figures

2.1 Intensity profile and its smoothings. (a) One-dimensional intensity profile. (b) The result of smoothing the profile in (a). (c) The result of additional smoothing of (a). (Reprinted from [9]).  

2.2 Image and its blurrings. (a) Original image. (b) The image blurred with a Gaussian $\sigma = 8$. (c) The image blurred with a Gaussian $\sigma = 4$. (Reprinted from [1]).  

2.3 A biological neuron includes three parts: dendrite, cell body and axon. (Reprinted from [16]).  

2.4 An artificial neuron.  

2.5 The artificial neuron and activation function for back-propagation network.  

2.6 The three-layer feed-forward network.  

3.1 (a) A 2-D image. (b) A slice of the 2-D image in (a). (c) The profile of the slice in (b), which resembles a step edge model.  

3.2 The feedward network for detecting 1-D step edge.  

3.3 (a) The non-noisy 1-D image profile. (b) The random noise profile. (c) The noisy 1-D image, which composed of (a) and (b). (d) The desired outputs and actual outputs of (c). (The vertical bars are the desired outputs, and the continuous plot is the actual output plot.)  

3.4 (a) The linear interpolation of intensity when $\text{ratio}_{a} < 0.5$. (b) The desired output representation of (a). (c) The linear interpolation of intensity when $\text{ratio}_{a} > 0.5$. (d) The desired output representation of (c). (e) The linear interpolation of intensity when $\text{ratio}_{a} = 0.5$. (f) The desired output representation of (e).  

3.5 The three-layer network architecture for edge detection.  

3.6 The window size of $7 \times 7$ is enough to reflect the properties.  

3.7 Directional primitives represented by the output units.
4.6 Common arrangements of lateral geniculate and cortical receptive fields.
(a) 'on' center geniculate receptive field. (b) 'off' center geniculate receptive field. (c) – (g) Various arrangements of simple cortical receptive fields. x, areas giving excitatory responses; Δ, areas giving inhibitory responses. (Reprinted from [15]).

4.7 (a) 2-D receptive fields profiles of simple cells in cat visual cortex, recorded by L. Palmer and J. Jones. (b) best-fitting 2-D Gabor filters. (c) residual error. (Reprinted from [33]).

4.8 (a). The isotropic weighting patterns, which are similar to non-oriented Gaussian filters. (b). The 3-D plot of the isotropic weighting patterns...

4.9 (a). The oriented weighting patterns for edges in the range of [0°, 22.5°].
(b). Some of the 3-D plot of the oriented weighting patterns...

4.10 (a). The oriented weighting patterns for edges in the range of [22.5°, 45°].
(b). The of the 3-D plot of the oriented weighting patterns...

4.11 (a). The oriented weighting patterns for edges in the range of [45°, 67.5°].
(b). Some of the 3-D plot of the oriented weighting patterns...

4.12 (a). The oriented weighting patterns for edges in the range of [67.5°, 90°].
(b). The of the 3-D plot of the oriented weighting patterns...

4.13 (a). The oriented weighting patterns for edges in the range of [90°, 112.5°].
(b). The of the 3-D plot of the oriented weighting patterns...

4.14 (a). The oriented weighting patterns for edges in the range of [112.5°, 135°].
(b). The of the 3-D plot of the oriented weighting patterns...

4.15 (a). The oriented weighting patterns for edges in the range of [135°, 157.5°].
(b). The of the 3-D plot of the oriented weighting patterns...

4.16 (a). The oriented weighting patterns for edges in the range of [157.5°, 180°].
(b). The of the 3-D plot of the oriented weighting patterns...

4.17 (a). The large-scale oriented filters. (b). The 3-D plot of the oriented weighting patterns.

5.1 Three possible architectures for detecting corner information.

5.2 Subjective contours. (Reprinted from [1]).
Acknowledgement

First, I would like to extend my special thanks to my supervisor Dr. David G. Lowe. Throughout my research, he carefully and patiently directed every step for me. Especially at times of difficulty, his encouragement and support contribute greatly to the work. My thanks also go to my second reader, Jim J. Little, for his valuable comments and advice.

Thanks to my husband, Yingchun, without whose support, this would never have happened.

Also, I would like to express my thanks to Jiping Lu, Xun Li and Yang Jian, they gave me very valuable discussion and help.

Last but not the least, a very special thanks to my parents. Even though far away from Vancouver, they have consistently provided me with the most thorough support and encouragement.
Chapter 1

Introduction

Edge detection plays an important role in many aspects of computer vision. In particular, it is very important for low-level vision.

People are good at detecting various scenes under a wide range of conditions, exhibiting very good visual capabilities. For example, people have no difficulty to detect blurred images, short bars, one-pixel wide thin bars, tiny blobs, etc. And also people will not confuse these details with unrelated noise. Compared with biological systems, the existing edge detectors have much less sensitivity and accuracy, although the design of these edge detectors are motivated in part by knowledge of biological vision [8]. In particular, these detectors take account of only a narrow range of spatial frequencies in the image and apply only a single specific scale filter to the image each time. If the chosen scale is too big, then many details, some of the real edges, are lost; if the scale is set too small, lots of unrelated details, including noise, are marked as real edges. Therefore, it will be hard to set a proper scale for an image which has a wide range of intensity changes. In addition, those edge detectors fail to make use of oriented filters. In this sense, they are not very sensitive to edge orientations. Although many attempts have been made to overcome these limitations and weak points, none of them have been successful enough to receive widespread use.

The purpose of this thesis is to overcome these limitations by exploring a new approach. The work is motivated by the analysis of the mammal primary visual cortex and the study of artificial neural networks.

The mammal primary visual cortex and its receptive fields have been extensively studied by physiologists. It has been found that early vision in mammals is performed in part by the simple cells of primary visual cortex [14]. Experiments made by D. H.
Hubel and T. N. Wiesel [15] have demonstrated that the receptive fields of the simple cells of visual cortex can be subdivided into distinct excitatory and inhibitory regions. There is summation within the separate excitatory and inhibitory parts, and there is antagonism between the excitatory and inhibitory regions. The spatial arrangements of the receptive fields differ profoundly from neuron to neuron. The varieties can be noticed in the respects of the axis orientations, the number of the subdivisions and the relative area occupied by each subdivision. When an edge stimulus is applied, the response of each neuron is different. Each of these neurons responds strongly only to edges at a particular range of orientations and spatial frequencies at a single location on the retina, with a response that is linear over a substantial range of input. Therefore, each neuron has its own optimal selective direction; for the maximum response the orientation of the edge is critical, and each neuron can be seen as an oriented linear filter.

In recent years, a great deal of research has been devoted to the design of optimal edge detectors [1] [2] [3] [4] [5] [7] [10]. Various methods have been tried from different points of view. Some of the most widely used existing edge detectors are those by Marr and Hildreth [3] and Canny [6]. Marr and Hildreth’s edge detector is based on the theory of edge detection which was proposed by Marr and Hildreth in 1979. According to this theory, a Gaussian filter is first applied to blur the image, then the Laplacian (rotation invariant operator) is applied. The zero-crossings of this function are extracted to form edges. In Canny’s edge detector, the second directional derivatives along the estimated gradient direction of an edge is calculated, and then the zero-crossing is marked and linked to get the edge description.

In order to study the role of the oriented linear filters in edge detection and to develop practical methods for combining filters at multiple scales and orientations, we have used the artificial neural network to perform the edge detection task, which is quite different from the conventional computational approach.

Artificial neural network models have received increased attention in recent years. Aimed at achieving human-like performance in tasks of the cognitive science domain, these models are composed of highly interconnected computing elements, whose structure is drawn from our current knowledge of biological systems. In particular, the neural network model did achieve lots of success in many areas, such as pattern recognition, pattern classification, image processing, computer vision and control.

We use the neural network model, not only because of its human visual cortex like structure, but also because of its human visual cortex like performance – multiple scaled
and oriented filter.

As for the learning algorithm, back-propagation [16] is used to train the network. Plaut and Hinton [17] have described the use of back-propagation for learning filters that are suitable for early processing of data in both speech and vision. Many successful examples, such as NetTalk [22], have been reported by using the back-propagation learning algorithm. Back-propagation can also achieve a high correctness rate when it is applied to classification problems.

Back-propagation allows us to train the connection weights in a feedforward network of arbitrary shape, where the energy surface for the descent is usually defined by the mean squared difference between the desired outputs and the actual outputs of the network.

To use the back-propagation algorithm, input-output pairs have to be designed very carefully. Initially, the 1-D case is tried to investigate the probability of solving this problem by using this algorithm. The input is a slice of image, which is the step function with noise. The desired output is the marker of intensity change. Then, with the successful results of 1-D case experiments, we come to the real 2-D case. Anti-aliasing [32] is used to generate various noise-polluted overlapped polygons and some special images. Simultaneously, the desired outputs are designed, including orientations and the edge marker.

As the hidden units in the back-propagation algorithm respond to a linear weighted sum of their inputs, they can be viewed as forming linear spatial filters through the modification of their input weights. These filters can then be combined in a non-linear fashion to produce the final edge detection output.

The resulting edges are finally thinned by a thinning algorithm. The location of each edge can be calculated to sub-pixel accuracy.

Compared with the results of Canny's edge detector, the results are very detailed, accurate in location, and sensitive to the orientations.

The thesis is divided up into five chapters. Chapter two presents a background and review of the areas related to the work. Some of the existing edge detectors, such as Marr and Hildreth's and Canny's are studied at first. Neural network methods are discussed in the rest of this chapter, from the human neuron anatomy to artificial neural network.

Chapter three describes the approach of the thesis. 1-D step function is discussed first, then 2-D case is presented in details, including the network architecture, the generation of training input-output pairs, the training scheme, the setting of learning pa-
rameters, the thinning and subpixel calculating algorithms.

Chapter four provides the testing results and examples examined on various real images. The results are compared with Canny's edge detection results. Some analysis of the results are also presented.

Also, in the second section of this chapter, we study the functionality of the hidden units, as well as the regularity of the weights pattern connected to the hidden units. The discussion includes its relationship with human visual cortex receptive fields.

Chapter five concludes the thesis with a discussion of the problems found using this approach, and any possible extensions and future work of the thesis.
Chapter 2

Background and Related Work

This chapter presents the necessary background information on edge detection and neural network learning. Section 2.1 gives an overview of the previous approaches used in edge detection and the disadvantages of these edge detectors. In section 2.2, neurons and neural networks are described from both anatomic and simulation points of view. Feed-forward neural network and back-propagation algorithm and its successful applications in other areas are introduced at the rest part of this chapter.

2.1 Previous Research on Edge Detection

For both biological systems and machines, vision begins with a large and unwieldy array of measurements of the amount of light reflected from surfaces in the environment. The goal of vision is to recover physical properties of objects in the scene, such as the location of object boundaries and the structure, color and texture of object surface, from the two dimensional image that is projected onto eye or camera. The goal proceeds in stages, with each stage producing increasingly more useful descriptions of image and then the scene. The first clue about the physical properties of the scene are provided by changes of intensity in the image.

Edges are curves in the image where rapid changes occur in brightness or in spatial derivatives of brightness. The changes in brightness that we are particularly interested in are the ones that mirror significant events on the surface being imaged. These might be placed where surface orientation changes discontinuously, where one object occludes another, where a cast shadow line appears, or where there is a discontinuity in surface reflectance properties. In each case, we hope to locate the discontinuity in image brightness, in order to learn something about the corresponding feature on the object
being imaged. Normally, noise in the brightness measurements limits the edge detector’s ability to uncover the true edge information.

Generally speaking, edge detection can be considered complementary to image segmentation, since edges can be used to break up images into regions that correspond to different surface. The importance of edge detection has led to extensive research both in computer and biological vision systems.

2.1.1 Marr and Hildreth’s Edge Detector

As described in [1] [3], Marr and Hildreth’s edge detection essentially incorporates three operations. First, the image intensities are smoothed with a two-dimensional Gaussian smoothing function,

\[ G(r) = \frac{1}{\sigma^2} e^{-r^2/2\sigma^2} \quad (2.1) \]

Second, the smoothed intensities are differentiated using the second-derivative operation. The second nondirectional operator that is used for detecting intensity changes is Laplacian operator \( \nabla^2 \),

\[ \nabla^2 f = \frac{\partial^2 f}{\partial^2 x} + \frac{\partial^2 f}{\partial^2 y} \quad (2.2) \]

Third, simple features in the result of this differential stage, zero-crossings (transitions between positive and negative values) are detected and described.

The smoothing operation serves two purposes. First, it reduces the effect of noise on the detection of intensity changes. Second, it sets the resolution or scale at which intensity changes are detected. The sampling and transduction of light by the eye or camera introduces spurious changes of light intensity that do not correspond to significant physical changes in the scene. Smoothing of the intensities can remove these minor fluctuations. Figure (2.1 (a)) shows a one-dimension intensity profile that is shown smoothed a little bit in figure (2.1 (b)).

Significant changes in the images can also occur at multiple resolutions. Consider, for example, a leopard’s coat. At a fine resolution rapid fluctuations of intensity might delineate the individual hairs of the coat, whereas at coarser resolution the intensity changes might delineate only the leopard’s spots. So, it is difficult to decide at what resolution the changes should be detected, and by what amount the image should be
smoothed. Figure (2.1 (c)) illustrates a more extensive smoothing of the intensity profile of figure (2.1 (a)), which preserves only the gross changes of intensity.

Figure 2.1: Intensity profile and its smoothings. (a) One-dimensional intensity profile. (b) The result of smoothing the profile in (a). (c) The result of additional smoothing of (a). (Reprinted from [9]).

Several arguments have been put forth in support of the use of Gaussian smoothing. Marr and Hildreth argued that the smoothing function should have both limited support in space and limited bandwidth in frequency [3]. In general terms, a limited support in space is important because the physical edges to be detected are spatially localized. A limited bandwidth in frequency provides a means of restricting the range of scales over which intensity changes are detected. The Gaussian function minimizes the product of bandwidths in space and frequency. Another reason that Gaussian function is popular for smoothing is that it is the only rotationally symmetric operator that is the product of two one-dimensional operators.

The differentiation operation accentuates intensity changes and transforms the image into a representation from which properties of these changes can be extracted more easily. A significant intensity change gives rise to a peak in the first derivative or a zero-crossing in the second derivative of the smoothed intensities. These peaks or zero crossings, can be detected straightforwardly.
The two-dimensional Gaussian smoothing function (Formula 2.1) and Laplacian operator (Formula 2.2) can be combined to yield the function $\nabla^2 G$ given by the expression

$$\nabla^2 G = \frac{1}{\sigma^2}(\frac{r^2}{\sigma^2} - 2)e^{-r^2/2\sigma^2} \quad (2.3)$$

where $r$ denotes the distance from the center of the operator and $\sigma$ is the standard deviation of the two-dimensional Gaussian. The $\nabla^2 G$ function is shaped something like a Mexican hat in two dimensions. The Laplacian is a nondirectional second-derivative operation; the elements in the output of the Laplacian that correspond to the location of intensity changes in the image are therefore the zero crossings. The zero-crossing contours were located by detecting the transitions between positive and negative values in the filtered image by scanning in the horizontal and vertical directions.

### 2.1.2 Canny's Edge Detector

In Canny’s thesis [5], Canny considered first the one-dimensional case of the edge detection problem with the traditional model of a step in white Gaussian noise.

Let the amplitude of the step be $A$, and let the variance of the input white noise be $n_0^2$. The input signal $I(x)$ can be represented by the step

$$I(x) = A \cdot u_1(x) + n(x) \quad (2.4)$$

with $n_0^2 = \langle n^2(x) \rangle$ for all $x$.

He assumed that detection was performed by convolving the noisy edge with a spatial antisymmetric function $f(x)$ and making edges at the maxima in the output $\theta(x_0)$ of this convolution:

$$\theta(x_0) = \int_{-\infty}^{+\infty} I(x) \cdot f(x_0 - x) dx \quad (2.5)$$

Trying to formulate precisely the criteria for effective edge detection, he set the following goals:

- **Good detection**

There should be a low probability of failing to detect a real edge point and low probability of falsely marking non-edge points. This criterion corresponds to maximizing signal-to-noise ratio (SNR), which is defined as the quotient of the response to the step only and the square root of the mean-squared noise response:
Finding the impulse response \( f(x) \) which maximizes \( \Sigma \) corresponds to finding the best operator for detection only.

- **Good location**
  
The points marked as edges by the operator should be as close as possible to the center of the true edge. This criterion corresponds to minimizing the variance \( \sigma^2 \) of the zero-crossing position or maximizing the localization criterion \( L \) defined as the reciprocal of \( \sigma \):

\[
L = \frac{A}{n_0} \cdot \frac{|f'(0)|}{[\int_{-\infty}^{+\infty} f^2(x)dx]^{1/2}} = \frac{A}{n_0} \cdot \lambda
\]  

Finding the impulse response \( f(x) \) which maximizes \( \lambda \) corresponds to finding the best operator for localization only.

- **One response to one edge**
  
The detector should not produce multiple outputs in response to a single edge. There is a need to limit the number of peaks in the response so there will be a low probability of declaring more than one edge. The distance between peaks in the noise response of \( f \), denoted \( x_{\text{max}} \), is set to be some fraction \( k \) of the operator width \( W \):

\[
x_{\text{max}} = k \cdot W = 2\pi[\int_{-\infty}^{+\infty} f^2(x)dx]^{1/2}
\]  

Having developed criteria for detection, localization, and limitation of the number of peaks, Canny combined them in a meaningful way: Maximize \( \Sigma \cdot \lambda \) (invariant under changes of scale or amplitude) under the constraint of the third criterion.

By expressing the criterion as a composite functional he found that this lead to solution \( f(x) \) such that

\[
2 \cdot f(x) - 2 \cdot \lambda_1 \cdot f'(x) + 2 \cdot \lambda_2 \cdot f''(x) + \lambda_3 = 0
\]  

with

\[
SNR = \frac{A}{n_0} \cdot \frac{\int_{-\infty}^{+\infty} f(x)dx}{[\int_{-\infty}^{+\infty} f^2(x)dx]^{1/2}} = \frac{A}{n_0} \cdot \sum
\]  

(2.6)
\[ \nabla = \lambda_2 - \frac{\lambda_1^2}{4} > 0 \] (2.10)

and where \( \alpha \) and \( \omega \) are real, such that:

\[ \alpha^2 - \omega^2 = \frac{\lambda_1}{2 \cdot \lambda_2} 4 \cdot \alpha^2 \cdot \omega^2 = \frac{-\lambda_1^2 + 4 \cdot \lambda_2}{4 \cdot \lambda_2^2} \] (2.11)

The general solution in the range \([0, W]\) may be written:

\[ f(x) = a_1 \cdot e^{\alpha \cdot x} \cdot \sin \omega \cdot x + a_2 \cdot e^{\alpha \cdot x} \cdot \cos \omega \cdot x + a_3 \cdot e^{-\alpha \cdot x} \cdot \sin \omega \cdot x + a_4 \cdot e^{-\alpha \cdot x} \cdot \cos \omega \cdot x + C \] (2.12)

Subject to the boundary conditions:

\[ f(0) = 0 \quad f(W) = 0 \quad f'(0) = S \quad f'(W) = 0 \] (2.13)

where \( S \) is an unknown constant equal to the slope of the function \( f(x) \) at the origin.

Since \( f(x) \) is antisymmetric, the above solution is extended to the range \([-W, +W]\) using \( f(x) = -f(-x) \). The four boundary conditions enable the quantities \( a_1 \) through \( a_4 \) to be determined.

After the image has been convolved with a symmetric Gaussian, the edge direction is estimated from the gradient of the smoothed image intensity surface. The gradient magnitude is then nonmaximum suppressed in the direction.

2.1.3 Problem with Multiple Scales

The analysis of intensity changes across multiple scales is a difficult problem that has not yet found a satisfactory solution. There is a clear need to detect intensity changes at multiple resolutions [2]. Important physical changes in the scene take place at different scales. Spatial filters that allow the description of fine detail in the intensity function generally miss coarser structures in the image, and those that allow the extraction of coarser features smooth out important detail. At all resolutions some of the detected features may not correspond to real physical changes in the scene. For example, at the finest resolutions some of the detected intensity changes may be a consequence of noise in the sensing process. At coarser resolution spurious image features might arise as a consequence of smoothing together nearby intensity changes.

Marr and Hildreth [3] explored the combination of zero crossing description that arise from convolving an image with \( \nabla^2 G \) operators of different size, see figure (2.2).
They suggested the use of spatial coincidence of zero crossings across scale as a means of indicating the presence of a real edge in the scene. Strong edges such as object boundaries often give rise to sharp intensity changes in the image that are detected across a range of scales and in roughly the same location in the image.

Canny [5] [6] used a different approach to combining description of intensity changes across multiple scales. Features were first detected at a set of discrete scales. The finest scale description was then used to predict the results of the next larger scale, assuming that the filter used to derive the larger scale description performs additional smoothing of the image.

Many other people also tried methods to overcome this problem, like Witkin [11], Poggio, Voorhees, and Yuille [12], Bergholm [13], but the problems of sorting out the relevant changes at each resolution and combining them into a representation that can be used effectively by later processes are extremely difficult and unsolved.

![Figure 2.2: Image and its blurrings. (a) Original image. (b) The image blurred with a Gaussian $\sigma = 8$. (c) The image blurred with a Gaussian $\sigma = 4$. (Reprinted from [1]).](image)

### 2.2 Neural Network

People have long been curious about how the brain works. The capabilities of the nervous system in performing certain tasks such as visual processings, speech are far more powerful than today’s most advanced computers. In addition to satisfying intellectual curiosity, it is hoped that by understanding how the brain works we will be able to create structure as powerful as, if not more powerful than, the brain.
2.2.1 The Basic Neuron Model

The neural network is the connection of neurons. A neuron consists of three parts: dendrite, cell body and axon. Dendrites receive impulse from other neurons, cell body sums these impulse, some inputs tending to excite the cell, others tending to inhibit it. When the cumulative excitation in the cell body exceeds a threshold, the cell "fires", and axon propagates the impulse. See figure (2.3).

![Figure 2.3: A biological neuron includes three parts: dendrite, cell body and axon. (Reprinted from [16]).](image)

The end of axon is synapse. In neural network, the neurons propagate information through synapse. People found that the foundation of memory is just occurred here. Two kinds of synapses - the excitatory and inhibitory synapse can make the network stable.

This is just the basic functional outline, nevertheless, most artificial neurons are designed based on the above mentioned simple characteristics.

In figure (2.4), a set of input label \((x_1, x_2, ..., x_n)\) is applied to the neuron. These inputs correspond to the signals into the synapses of a biological neuron. Each signal \(x_i\) is multiplied by an associated weight \(w_i \in w_1, w_2, ..., w_n\), before it is applied to the summation block. Each weight corresponds to the "strength" of a single biological synapse. the summation block, corresponding roughly to the biological cell body, adds up all the weighted inputs, producing an output \(\epsilon = \sum^n w_i x_i\). \(\epsilon\) is then compared with a threshold \(t\). The comparing result is usually further processed by an activation function to produce the neuron's output. The activation function may be a sigmoid function, a simple linear function, a threshold-logic function, or a function which more accurately simulate the nonlinear transfer characteristic of the biological neuron and permits more general network functions.

Figure (2.5) illustrates the neuron used as the fundamental building block for back-propagation network. A set of inputs is applied, either from the outside or from a
previous layer. Each of these is multiplied by a weight, and the products are summed. This summation of products is termed $NET$ and must be calculated for each neuron in the network. After $NET$ is calculated, an activation function $F$ is applied to modify it, thereby producing the signal $OUT$.

The function: $OUT = 1/(1 + e^{-NET})$ is called sigmoid function, which is desirable in that it has a simple derivative, a fact we use in implementing the back-propagation algorithm.

$$\frac{\partial OUT}{\partial NET} = OUT(1 - OUT)$$

sometimes called a logistic, or simply a squashing function, the sigmoid compresses the range of $NET$ so that $OUT$ lies between zero and one. Multilayer networks have greater representational power than single-layer networks only if a nonlinearity is introduced. The squashing function produces the needed nonlinearity.
There are many functions that might be used; the back-propagation algorithm requires only that the function be everywhere differentiable. The sigmoid satisfies this requirement. It has the additional advantage of providing a form of automatic gain control. For small signals (NET near zero) the slope of the input-output curve is steep, producing high gain. As the magnitude of the signal becomes greater, the gain decreases. In this way, large signals can be accommodated by the network without saturation, while small signals are allowed to pass through without excessive attenuation.

2.2.2 The Neural Network Models

Although a single neuron can perform certain pattern detection functions, the connected neurons which forms the neural network can be more powerful [24].

Neural networks are interconnected groups of living cells called neurons. The neurons are connected by synapses which allow one neuron to excite or another. The human brain is an example of a large neural work which is capable of performing many complex cognitive tasks. Various neural network models have been designed and used in applications.

Network topologies can be divided into two broad categories called feed-forward and recurrent. Here, we mainly introduce the feedforward network.

In feedforward networks, neurons are arranged in layers. There are connections between neurons in different layers, but no connection between neurons in the same layer. The connection is unidirectional. The output of a feedward network at time \( k \) depends only on the network input at time \( (k-1) \). In other words, the output of one particular neuron can never contribute to the input of that same neuron, either directly or indirectly. Figure (2.6) shows a three-layer feed-forward network.

Many neural network models belong to this class, examples are Perceptron, back-propagation network, self-organizing feature map, counter-propagation network, Neocognitron, and functional-link network. In the following, the back-propagation network learning algorithm is described since it is the most popular one in neural network application, also it is just the network we employed in this thesis.

2.2.3 Back-propagation and its Applications

Back-propagation [16] developed by Rumelhart and his PDP group can be considered a generation of the perceptron learning procedure for multilayer nonlinear networks of
neuronlike computing elements and answers the basic objections of Minsky and Papert [26] to such learning mechanism.

The basic idea is to change the weights between the units in such a way as to reduce the error in the output. An input pattern is presented to the network, and activation is propagated forward through the network to the output units. The correct output pattern is provided to the output units in the form of a teaching signal. It is clear how output units should change their weights to reduce the error. If their value is too low, they should raise the weights on active input lines. If their value is too high, they should lower the weights. This will tend to make them attain the proper value in the same situation in the future. What is difficult to determine is how the hidden units should change their weights—there is no explicit teacher for them. Back-propagation provides a rule for propagating an error signal back through the network from the output units, which tells the hidden units which way to change their weights. The surprising thing is that the rule for changing the weights is purely local one, in the sense that every unit can find out its error through the connections that already existed in the network.

The weight-changing rule is derived by defining an error measure—the mean square error of the output—and taking the partial derivative of this with respect to the weights. Intuitively, we can think of the error as a surface over the space of possible weights. Taking the partial derivative tells us which way to move in weight space in order to go downhill fastest. Because this is a gradient descent algorithm for reducing the error, it
is subject to the major problem associated with gradient descent procedures, i.e. **local minima** in the error surface. However, experience has shown that we very rarely run into the problem of local minima in networks with many weights.

In some cases, units in hidden layers develop interesting response properties, that is, they have become feature detectors for different important aspects of input.

The back-propagation is an extremely useful algorithm because it is simple and robust and does well in practice in a wide range of areas, including classification [29], pattern recognition [30], speech [28], especially image processing [31] and early vision.

NEC in Japan has announced recently that it has applied back-propagation to a new optical-character-recognition system, thereby improving accuracy to over 99% [27].

Sejnowski and Rosenberg [22] produced a success with Nettalk, a system that converted printed English text into highly intelligible speech.

Burr (1987) has used back-propagation in machine recognition of handwritten English words. The characters are normalized for size, are placed on a grid, and projections are made of the lines through the squares of the grid. These projections then form the inputs to a back-propagation network. He reports accuracies of 99.7% when used with a dictionary filter.

Cottrell, Munro, and Zipser [31] report a successful image compression application in which images were represented with one bit per pixel, an eightfold improvement over the input data.

In this thesis, we use a back-propagation network to perform the edge detection task.

### 2.2.4 Plaut and Hinton's Paper

In paper [17] (1987), Plaut and Hinton use back-propagation to develop a set of filters for difficult problems like speech recognition and vision tasks.

This set of filters are good at discriminating between rather similar signals in the presence of a lot of noise, tuned to the critical difference accurately. The filters cover the range of possible frequencies and onset times, and when several different filters fit quite well, their outputs can be correctly weighted to give the right answer. From the weighting patterns connected to the hidden units, it can be noticed that each filter covers several different cases and that each case is covered by several different filters. Several filters can cooperate to get the correct answers. The set of filters form an "ecology" in which each one fills a niche that is left by the others. The optimal value of each weight
depends on the value of every other weight.

The back-propagation learning algorithm can be viewed as a numerical method of solving the analytically intractable design problem. Therefore, it is quite useful for exploring the space of possible filter designs.

The work of this thesis is inspired by Plaut and Hinton’s paper. In Plaut and Hinton’s paper, a set of linear filters are also developed by using back-propagation to perform the early vision task. The result is quite initial. Based on Plaut and Hinton’s basic idea, we design an edge detection model, which is more complete and practical.
Chapter 3

A Learning Approach for Edge Detection

In this chapter, a detailed description of the new edge detector is given. The detector of 1-D edges, which are a step function is first briefly introduced in Section 3.1. Then the 2-D edge detector is discussed in the rest of this chapter, including the neural network architecture, the input-output pairs generation, the training process, and finally, the thinning method and the subpixel accuracy calculation algorithm.

3.1 1-D Edge Detection

The basic design problem is illustrated in figure (3.1). We are trying to detect a step edge which is bathed in noise, figure (3.1c). The objective is to find a spatial filter using neural network, which gives the strong response or best output on the changes of intensity.

The feedward neural network architecture used here for 1-D step edge is schematically shown in figure (3.2), each of the circles represents a neuron. The network consists of three layers: input layer (9 neurons), hidden layer (10 neurons) and output layer (1 neuron).

For supervised learning, training data has to be prepared, it includes the input patterns and the corresponding desired outputs.

The construction of the image data is done in this way: a segment of 1-D non-noisy profile which contains a variety of step edges is generated, such as low contrast edges, high contrast edges, near edges, blurred edges, sharp edges, etc. Then, random noise is prepared; finally, the step edges with random noise is composed by putting the non-noisy
Figure 3.1: (a) A 2-D image. (b) A slice of the 2-D image in (a). (c) The profile of the slice in (b), which resembles a step edge model.
step edges and the noise together. The desired outputs are made according to the step edges, see figure (3.3).

The linear interpolation is used to sample the step edges and to produce the corresponding desired outputs. Suppose that a step edge is just falling on the pixel $b$, which is between pixel $a$ and pixel $c$. We know the intensities of pixel $a$ and pixel $c$. The distance ratio from the edge to $a$ is $\text{ratio}_a$, so the distance ratio from the edge to $c$, $\text{ratio}_c$, is $\text{ratio}_c = 1 - \text{ratio}_a$. The intensity of $b$ can be determined by this formula:

$$\text{intensity}_b = \text{intensity}_a \cdot \text{ratio}_a + \text{intensity}_c \cdot \text{ratio}_c$$

The desired outputs are designed as real values in the range of $[0.1, 0.9]$ to denote the edge occurrence certainty. So, the desired outputs of $a$, $b$ and $c$:

$$\text{desired}_a = \begin{cases} 0.1 + 0.8 \times (0.5 - \text{ratio}_a) & \text{if } \text{ratio}_a <= 0.5 \\ 0.1 & \text{otherwise} \end{cases}$$

$$\text{desired}_c = \begin{cases} 0.1 \quad & \text{if } \text{ratio}_a <= 0.5 \\ 0.1 + 0.8 \times (0.5 - \text{ratio}_a) & \text{otherwise} \end{cases}$$

$$\text{desired}_b = \begin{cases} 1 - \text{desired}_a & \text{if } \text{ratio}_a <= 0.5 \\ 1 - \text{desired}_c & \text{otherwise} \end{cases}$$

The detailed explanations are shown in figure (3.4). The desired outputs are the inverse of the distance from the edge to the center of the nearby pixels.

A nine-pixel running window of is used to scan the input segment. The center pixel is checked during the training.
Figure 3.3: (a) The non-noisy 1-D image profile. (b) The random noise profile. (c) The noisy 1-D image, which composed of (a) and (b). (d) The desired outputs and actual outputs of (c). (The vertical bars are the desired outputs, and the continuous plot is the actual output plot.)
The back-propagation algorithm is used to train the network, which will be discussed in detail later. After a period of training, the testing result is shown in figure (3.3(d)). The vertical bars are the desired outputs of the step function, and the continuous plot is the actual output plot. From the figure, we can see the responses of the edges are correct and accurate, no false edges, no missing edges. The actual outputs are very close to the desired outputs.

Figure 3.4: (a) The linear interpolation of intensity when \( \text{ratio}_a < 0.5 \). (b) The desired output representation of (a). (c) The linear interpolation of intensity when \( \text{ratio}_a > 0.5 \). (d) The desired output representation of (c). (e) The linear interpolation of intensity when \( \text{ratio}_a = 0.5 \). (f) The desired output representation of (e).
3.2 The Neural Network Architecture to Perform 2-D Edge Detection

In this section, we will give the detailed discussion of the network configurations for 2-D edge detection. Figure (3.5) shows the three-layer network for edge detection. The network consists of $7 \times 7$ neurons in the input layer, a number of units in the hidden layer and 17 neurons in the output layer. Each layer is fully connected by the weights to next layer.

![Diagram of a three-layer neural network for edge detection.](image)

Figure 3.5: The three-layer network architecture for edge detection.

The first layer receives inputs from a 2-D patch of $7 \times 7$ pixels. The center pixel is checked. The reason that we choose $7 \times 7$ as the window size is that we know each pixel’s properties are significantly influenced by its nearby pixels. For example, with junctions, a $7 \times 7$ window is enough to let the network see T-junctions, V-junctions, and also the certainties of the center pixel falling on one edge. See figure (3.6). But sometimes, an even larger window maybe necessary in some cases, such as very noisy images or very blurred images.

The second layer is necessary for our task. The learning is mainly performed by this layer. Each neuron can be seen as a function or filter to map the input patterns to the output. In this case, multiple hidden units can be seen as an adaptive multi-scaled filter to interpret the input image to the output layer. We do experiments using different number of hidden units, such as 20, 40, 70 and 80. The more hidden units are used, the longer time the network needs to train. By comparing the results, we find that hidden
The window size of $7 \times 7$ is enough to reflect the properties. A layer with 70 units is appropriate for our problem.

After filtered by the hidden layer, the outputs of the hidden units are fed to the output layer to extract the features that we want the network to learn. 17 neurons represent 17 visual primitives, including the directional information (see figure (3.7)) and the intensity change information.

3.3 **Input-Output Training Data Generation**

Input patterns and the desired outputs are generated for the back-propagation network. The full process for preparing data is illustrated in figure (3.8) and figure (3.9).

3.3.1 **Input Data Generation**

The sets of image training data are produced from two points of view: trying to cover the features of real images, and trying to make the generating process very simple.

As $7 \times 7$ window size is used to scan the image, the curve of small curvature can almost be regarded as a straight line, and the curve of very big curvature can be treated...
Figure 3.7: Directional primitives represented by the output units.
Figure 3.8: The general procedure for generating the input training data.
Figure 3.9: The general procedure for generating the desired output data.
as a V-junction inside the window. Therefore, various kinds of polygon images are synthesized by computer.

1. **Overlapped polygons with sharp edges**

   Every polygon is drawn separately:
   
   - Select vertices of polygon at random locations, such as four vertices
   - Connect the boundary according to the polygon vertexes
   - Fill the inner area of the enclosed polygon with required intensity

   Several polygons are produced at the same time, then overlap them together. At first, the $250 \times 250$ image is generated, then we average every $5 \times 5$ square of image to achieve proper precision of intensities of a $50 \times 50$ image. Noise is added to the image randomly. See figure (3.10).

   The intensity of each one can be selected either by normal contrast or by low contrast, see figure (3.11). As low contrast images are more difficult for the network than those normal contrast images, a special treatment for low contrast images is needed. We design some low contrast images purposely. The ratio between normal contrast values and low contrast values is 8 to 2. For low contrast values, the intensities difference is between 5 and 20; for normal contrast values, the intensities difference is between 5 to 250.

   Something has to be done to prevent polygons with equal intensities. We generate several random numbers in the range of $[0, 255]$ without repetition according to the requirements, then fill the polygons with those numbers.

2. **Overlapped polygons with blurred edges**

   Because blurred images are also a kind of difficulty for the network, Gaussian masks are applied to the image to get the blurred results. Two Gaussian masks are used, with one having more blurring and another having less blurring. The two masks are illustrated in figure (3.12). The blurred results is shown in figure (3.13). Noise is also needed to add to the blurred image at the final stage.

3. **Some special kinds of image**
Figure 3.10: The $250 \times 250$ image and the $50 \times 50$ image after averaging.
Figure 3.11: The general overlapped polygons with sharp edges.

Figure 3.12: The two blurring masks.

Figure 3.13: The general overlapped blurred edges polygons.
Overlapped polygons can not cover all the special cases of images, we need some special kinds of polygons, such as, one-pixel wide thin lines, thin bars, strips, etc, see figure (3.14).

![Figure 3.14: Some special kinds of features: one-pixel thin line, strips, thin bars, etc.](image)

A $7 \times 7$ running window is used to scan the whole image, see figure (3.15).

![Figure 3.15: A $7 \times 7$ window is used to scan the whole image, selecting patterns from it.](image)

There are several criteria to select a window as an input pattern of the network to get the efficiency of training, see figure (3.16). The probability of selecting a window is determined by the difficulty and the chance of the occurrence of the window.

- **Case 1**: If the center pixel of the window is the vertex of a corner, either V-corner or T-corner, then the probability of choosing the window is 80%. The reason is that corner occurrence is relatively rare, and corner is hard to learn, so the majority is selected.
Case 2: If the center pixel of the window is on an edge, then the probability of choosing the window is 50%.

Case 3: If there is an edge passing through the window, but one pixel away from the center, then the probability of choosing this window is 30%.

Case 4: If there is an edge passing through the window, but two pixels away from the center, then the probability of choosing the window is 20%.

Case 5: If no edge and no corner in the window, then the probability of choosing the window is 10%.

Figure 3.16: The criteria of selecting a window.

If a window is chosen, the values of the vector are normalized to the range of $[0.1, 0.9]$ by the formula:

\[
input = \frac{intensity}{256} \cdot 0.8 + 0.1
\]

3.3.2 Desired Output Design

The range of orientations is divided into eight parts, with each part spanning $22.5^\circ$. 16 units are responsible for 8 directions; and 1 unit for general information. Two units are for one direction to distinguish the intensity changing from dark to white and from white to dark. We consider two factors to produce the output: the orientation and the distance
to the center of the pixel. The value of each desired output unit ranges from 0.1 to 0.9 continuously. The value of each direction output unit represents the distance from the center of the pixel to the edge. The value of the desired direction output value increases linearly as the center of the pixel gets closer to the edge, and decreases linearly as the center of pixel gets farther away from the edge. Similarly, the value of each pixel linearly interpolates between neighboring orientation outputs. The example (figure (3.17)) is given below:

Figure 3.17: The angle calculation: (a) The plane is divided into 8 parts (b) An edge falling between direction_a and direction_b. (c) The distance calculation.

Suppose the edge falls between direction_a and direction_b, angle_a + angle_b = 22.5°. If we let ratio_a = angle_a/22.5°, then ratio_b = 1 - ratio_a.

Suppose the edge function is:

\[
(y_1 - y_0) \cdot x - (x_1 - x_0) \cdot y + x_1 \cdot y_0 + x_0 \cdot y_1 = 0
\]

The distance Distance from the center (C_x, C_y) of the pixel to the edge:

\[
Distance = \frac{(y_1 - y_0) \cdot C_x - (x_1 - x_0) \cdot C_y + x_1 y_0 + x_0 y_1}{\sqrt{(y_1 - y_0)^2 + (x_1 - x_0)^2}}
\]

If the intensity changes from dark to white, then the desired output of units \((2 \cdot a) \text{mod} 16\) and \((2 \cdot b) \text{mod} 16\) can be determined:

\[
\text{Desired}_{(2 \cdot a) \text{mod} 16} = (1 - \text{ratio}_a) \cdot (1 - Distance)
\]

\[
\text{Desired}_{(2 \cdot b) \text{mod} 16} = (1 - \text{ratio}_b) \cdot (1 - Distance)
\]
But if the intensity changes from white to dark, then the desired output of units \((2 \cdot a + 1)\mod 16\) and \((2 \cdot b + 1)\mod 16\) can be determined:

\[
\text{Desired}_{(2 \cdot a + 1)\mod 16} = (1 - \text{ratio}_a) \cdot (1 - \text{Distance})
\]

\[
\text{Desired}_{(2 \cdot b + 1)\mod 16} = (1 - \text{ratio}_b) \cdot (1 - \text{Distance})
\]

One unit of the output layer is for detecting the intensity change, which can be calculated from the 16 directional values.

In this way, we can represent two edges of one-pixel thin line, see figure (3.18).

![Figure 3.18: A one-pixel thin line and its desired outputs](image)

Figure 3.18: A one-pixel thin line and its desired outputs

Figure (3.19) shows a raw image and its 17 desired outputs.

### 3.4 Training Algorithm and Training Process

#### 3.4.1 Back-propagation Algorithm

The aim of the learning procedure [23] is to find a set of weights such that, when the network is presented with each input vector, the output vector produced by the network is the same as (ideally) or sufficiently close to the desired output vector. That means the error \(E\) made by comparing the actual and desired output vector should be least.

\[
E = \frac{1}{2} \cdot \sum_p \sum_j (o_{p,j} - t_{p,j})^2
\]  

(3.1)

where \(p\) is an index over patterns, \(j\) is an index over output units. \(o_{p,j}\) is the actual output, and \(t_{i,j}\) is the desired output.
Figure 3.19: A raw image and its 17 desired outputs.
So the network runs in two stages: a forward pass in which the state of each unit in
the network is set, and a backward pass in which the learning procedure operates.

During the forward pass, an input vector is presented sequentially, starting at the
bottom and working upwards. The states of units in each successive layer are determined
in parallel. The forward pass is complete once the states of the output units have been
determined.

The backward pass starts with the output units at the top of the network and suc­
cessively works down through the layers, “back-propagating” error derivatives to each
weight in the network.

The following is the steps of standard back-propagation:

1. Let $A$ be the number of units in the input layer (determined by the length of the
   training inputs), and $C$ be the number of units in the output layer. Choose $B$
   the number of units in the hidden layer. The hidden and input layers will each
   have an extra unit used for thresholding; therefore, the units in these layers will
   be indexed by the ranges $(0...B)$ and $(0...A)$. We will denote the activations of the
   units in the input layer by $i_i$, in the hidden layer by $h_i$, and in the output layer by
   $o_i$.

2. Choose an input-output pair. Suppose the input vector is $x_i$ and the target output
   vector is $t_i$. Initialize the activations of the input layer units according to input
   layer:

   $i_i = x_i$

   for $j = 1...A$.

3. Propagate the activations from the units in the input layer to the units in the hid­
gen layer. Activations of hidden layer units are computed according to activation
   function.

   $h_i = \frac{1}{1 + exp(-\sum_{k=0}^{A} w_{ij}i_k)}$

   for all $j = 1...B$.

   The expression $exp(x)$ means $e$ raised to the $x$ power. Notice that one of the values
   of $k$ is 0. $w_{0j}$ is the threshold weight for hidden unit $j$. Consider $i_0$ always to have
   the value 1.
4. Propagate the activations from the units in the hidden layer to units in the output layer.

\[ o_j = \frac{1}{1 + exp(- \sum_{k=0}^{B} w_{kj}h_k)} \]

for all \( j = 1...C \).

Again, the threshold weight \( w_{0j} \) for output unit \( j \) takes part in the weighted summation. Consider \( h_0 \) to be 1.

5. Compute the error of the units in the output layer, denoted by \( \eta_j O \),

\[ \eta_j O = o_j(1 - o_j)(t_j - o_j) \]

for all \( j = 1...C \).

6. Compute the error of the units in the hidden layer, denoted by \( \eta_j H \),

\[ \eta_j H = h_j(1 - h_j) \sum_{k=1}^{C} \eta_k O_{w_{jk}} \]

for all \( j = 1...B \).

7. Adjust the weights between the hidden layer and output layer. We denote the learning rate as \( \eta \).

\[ \Delta w_{kj}(t + 1) = l\eta_j O_{h_k} + \alpha \Delta w_{ij}(t) \]

for all \( j = 1...C, k = 0...B \).

This weight adjustment includes the threshold weight \( w_{0j} \). Remember that \( h_0 \) is 1.

8. Adjust the weights between the input layer and the hidden layer

\[ \Delta w_{kj}(t + 1) = l\eta_j H_{h_k} + \alpha \Delta w_{ij}(t) \]

for all \( j = 0...B, k = 0...A \).

Once again, the threshold weight \( w_{0j} \) is also adjusted for each unit \( j \), \( i_0 \) is always 1.

3.4.2 Rescaling Some Variables

In the back-propagation, the chain rule of differentiation introduces the factor $o \cdot (1 - o)$, $o$ is the output of each node. The factor $o \cdot (1 - o)$ occurs once in the output layer, twice on the first hidden layer, ..., and so on, until the input layer is reached. Also, it should be noted that can not exceed $1/4, 1/16, ...$ at the various layers, causing the gradient vector to differ radically in magnitude. The compensatory rescaling is need for the partial derivative. The rescaling $4, ...$ is applied to the sequential layers in our implementation. Comparing with standard back-propagation, back-propagation with rescaling variables [20] is faster, the training time plot is shown in figure (3.20).

![BP & RBP Error plot](image)

Figure 3.20: The error plot of standard back-propagation and improved back-propagation with rescaling some variables. (dotted line: BP; solid line: rescaling BP).

3.4.3 Training Process

In our training process, we use one.epoch, which means the new patterns are generated each time, no input pattern is learned repeatedly, and the input patterns are picked randomly from the new generated image. In this case, we have plenty of training data. The learning error is corrected whenever each pattern is presented. The learning process stops when the learning error is sufficiently small, see figure (3.21).

Several learning parameters have to be chosen very carefully [25]; a small difference in these parameters can lead to a large difference in learning times. There are three parameters that must be set: $\epsilon$, the learning rate; $\alpha$, the momentum factor; and $\gamma$, the range of the random initial weights. Plaut, Nowlan, and Hinton suggest that the $\epsilon$ value should be inversely proportional to the fan-in of the neurons in the network, something like 0.05 to get big learning steps after the network has begun to move in a consistent direction through weight space; then, $\epsilon$ is tuned gently according to the epoch errors;
Figure 3.21: The training process is one epoch.
finally, $\epsilon$ is set to be very small again to learn fine details. The shape of learning rate curve is plotted in figure (3.22). $\alpha$ is chosen to be 0.9 through the whole training. $\gamma$ is initially randomly set from 0.3 to -0.3. When the $|\gamma|$ is initially set to be bigger than 1, it will need a much longer time to train; when $|\gamma|$ is initially set very big (such as around 4), the network will explode and never converge.

![Learning Rate Curve](image)

Figure 3.22: The change of learning rate during the learning process.

The training is performed on a MIPS RC3260 computer. Usually a number of hours are needed to get the training result. We do experiments using different number of hidden units, such as 40, 50, 70, and 80. The reason that we can use so many hidden units is that plenty of input patterns can be generated so that the network can be never over trained. Overtraining means the network just remembers some input patterns but does not really learn the features.

### 3.5 Thinning Algorithm and Subpixel Calculation

Thinning and subpixel calculation is an important step for edge detection. This step involves computing the accurate position of an edge and removing points from the raw edges.

The input of this procedure is 17 output images, representing 16 directions and 1 general edge marker, see figure (3.23).

The 17 images are scanned simultaneously. Whenever a pixel is met whose response is bigger than the threshold in some direction, it is denoted as the StartPoint. The accurate direction can be calculated by linearly interpolating between this directional value and
16 directional images
1 edge marker

Thinning
Accurating

thin edges
the direction of
each edge

Figure 3.23: The input and output of thinning and subpixel calculation algorithm.

the neighboring directional value. The edge can be traced along this direction from the StartPoint. When tracing the edge, accurate location can be computed by linearly interpolating the value of this pixel and the value of nearby pixel. In the meantime, non-maximum suppression is done repeatedly in the direction perpendicular to the edge.

The general procedure is shown in figure (3.24). Several issues labeled in the figure are explained here:

1. Calculate the accurate direction

The accurate direction of the edge is calculated based on direction response in \( D_i, D_{i-2} \) or \( D_{i+2} \), see figure (3.25).

2. Calculate the accurate location

The accurate location of the edge is calculated by linearly interpolating the response \( P_0 \) in direction \( D_i \) with nearby pixel's response in direction \( D_i \), see figure (3.26). If the angle of the edge is quite steep which is in the range of \([45°, 135°]\), the linear interpolation is calculated between the current pixel and the left or right pixel depending on which value is bigger; if the angle of the edge is relatively flat which is in the range of \([0°, 45°]\) or \([135°, 180°]\), the interpolation is done between the current pixel and the upper or lower pixel depending on which one is bigger.

3. Non-maximum suppression

Non-maximum suppression is done after the linear interpolation. The suppression is performed either horizontally or vertically according to the angle of the edge. If the angle is in the range of \([45°, 135°]\), then left pixels and right pixels are iteratively suppressed if the value keeps decreasing, until another peak is met; if the angle is in the range of \([0°, 45°]\) or \([135°, 180°]\), upper and lower pixels are suppressed as long as the value keeps decreasing.
Figure 3.24: The thinning and subpixel calculation algorithm.
IF (Di-2 > Di+2)
{
    /* linearly interpolate Di and Di-2 */
    Direction = integer(Di / 2 * 22.5 - 22.5 *(Di-2 / (Di + Di-2) + 180) % 180 ;
}
ELSE
{
    /* linearly interpolate Di and Di+2 */
    Direction = integer(Di / 2 * 22.5 + 22.5 *(Di+2 / (Di + Di+2) + 180) % 180 ;
}

Figure 3.25: The calculation of accurate direction.

Figure 3.26: The calculation of accurate location.
In addition, we suppress non-maximum directionally. If current direction index is $D_i$, we keep suppressing in both side starting from $D_i$, which means we will suppress $D_{(i-2+\text{DirectionNum})\%\text{DirectionNum}}, D_{(i-4+\text{DirectionNum})\%\text{DirectionNum}}, \ldots$, as long as it is decreasing; also suppress $D_{(i+2)\%\text{DirectionNum}}, D_{(i+4)\%\text{DirectionNum}}, \ldots$, until another peak is met, see figure (3.27). The directional suppression is done at each pixel that is suppressed in left and right directions. All related pixels must be suppressed.

![Diagram of non-maximum suppression](image)

Figure 3.27: Non-maximum suppression.

4. **Next pixel selections**

The next pixel is first selected along the direction of the edge, which means we keep tracing along the edge until the value of the candidate is less than the threshold. We choose the next one among the possible pixels according to the direction of the edge, see figure (3.28).

When the value of the selected pixel is less than the threshold using the above method, another selecting method is used. We select the candidate among all neighboring pixels (we consider 8-connected) no matter what direction it is. In this way, we can trace some curves, blobs continuously, not only the lines, see
Candadite is selected from pixel 1, 2, 3

Figure 3.28: Select the next pixel along the edge.

figure (3.29).

8-Connection

Candadite is selected from pixel 1, 2, 3, 4, 5, 6, 7, 8

Figure 3.29: Select the next pixel from neighboring pixels.

The thinning results of some examples from real images are given in next chapter.
Chapter 4

Results and Hidden Units Analysis

In this chapter, we will present the experimental results and compare the results with Canny’s. The results are given in two stages: neural network testing results, including the directional results and general edge result; the thinning and subpixel calculating results. In the second part of this chapter, we will discuss and analyze the functionality of hidden units and their receptive fields.

4.1 Experimental results

After training 100 $50 \times 50$ images for 200 epochs, which took about 40 hours running on a MIPS RC3260 computer, the weights are usually converged to stable values. We use real images to test the neural network training results. The results are finally thinned by the thinning algorithm.

The first example is UBC buildings image taken with a film camera (figure (4.1)). We give 17 neural network testing result images, including 1 general edge image and 16 orientation images. Each orientation image shows all the edges of a particular orientation in the image. The edge marking image is the sum of all the 16 orientation image.

From the orientation images we can see that each orientation image contains all the edges within a narrow range of orientations. Some of the edge responses are very strong (the relatively dark lines), which means their orientations are close to the center of the orientation range and their locations are close to the center of a pixel. Also, we can see that some of the very weak responses (the white lines that are even lighter than the background) are just perpendicular to the dark lines. The explanation is very clear: when a neuron is excited by the edges of a particular orientation, it will be inhibited by the edges that are perpendicular to the orientation.
From the general edge image, we notice that the results from the neural network are significantly more detailed and more precise in locating edges than Canny's edge detector. In Canny's edge detector, we use $\sigma = 0.7$ and $\sigma = 1.5$ to detect edges of an image. Although the edge result of $\sigma = 0.7$ is more detail than the result of $\sigma = 1.5$, it is quite sensitive to noise. But the result of $\sigma = 1.5$ miss some real edges. So it is hard to set a proper $\sigma$ of Canny's edge detector to detect details but not sensitive to noise.

(a). UBC Buildings image.
(b). Neural network detecting result: the sum of 16 orientations.
(c). 16 orientation images detected by neural network.
(d). Neural network result after thinning.

(e). Canny's detecting result: $\sigma = 0.7$.  
(f). Canny's detecting result: $\sigma = 1.5$.

Figure 4.1: The experimental results of UBC buildings image.
The second example (figure (4.2)) is the image of a person holding a box which is taken by a TV camera. There are some blurred edges and curves in this image. The test shows that the results from the neural network are less sensitive to noise and less distorted than Canny’s edge detector.

The third and fourth experiment images are synthesized by computer. They are 3-pixel wide and 2-pixel wide black and white bar strips. First, the non-noisy black and white bar strips are generated with the intensity changed gradually brighter from left to right. Second, we generate the noise, with the amount of noise increased gradually from top to bottom. Then, we put the bar strips and the noise together to produce the synthesized image. See figure (4.3).

The testing results (figure (4.4), figure (4.5)) are very impressive, with the neural network results are superior to the Canny’s results. For Canny’s detecting results, both for $\sigma = 0.7$ and $\sigma = 1.5$, it is difficult to link the broken short segments into straight edges appeared in the raw images. But the results of neural network are much better.

(a). A raw image of a person holding a box.
Figure 4.2: The experimental results of a person holding a box.
(a). Image without noise.

(b). Noise increasing from top to bottom.

(c). Image composed by (a) and (b).

Figure 4.3: The synthesis procedure of a image.
4.2 Hidden Units Analysis

We claim that our method has developed a set of linear filters for various kinds of edges and curves. These multiple scaled, oriented filters bear a close qualitative resemblance to visual neuron receptive fields found in mammalian primary visual cortex.

4.2.1 Biological Visual Primitives

The primary visual cortex of a mammal contains several populations of neurons, some linear and some nonlinear, with selectivities for a variety of visual stimulus attributes. These include location in visual space, size, orientation, motion, color, stereoscopic depth, spatial frequency, and etc. Perhaps the most remarkable among these is orientation selectivity, which imparts to individual neurons a strong dependency of their firing rate on the orientation of an edge or blob of light in their receptive field. Moreover, assemblies of neurons are organized into “columns” which share the same orientation preference, and on a still larger scale, these columns reveal a functional “sequence regularity” of systematic shifts in cells’ preferred orientation. This sequence regularity of columnar orientation preference is one of the most obvious features of visual cortical architecture discovered to date, and it clearly plays a crucial, if as yet unspecified, role in the logic of the brain’s representation of visual world.

Early vision in mammal is performed in part by the cells of primary visual cortex with simple receptive fields [19]. The receptive fields of these cells can be subdivided into distinct excitatory and inhibitory regions, there is summation within the separate excitatory and inhibitory parts, there is antagonism between excitatory and inhibitory regions and it is possible to predict response to stationary or moving spots of various shapes from a map of the excitatory and inhibitory areas.

The cells with simple receptive fields differ profoundly in the spatial arrangements of excitatory and inhibitory regions. Some of these cells, such as retinal ganglion and geniculate cells, have one or other of the concentric forms shown in figure (4.6 (a) (b)). (Excitatory areas are indicated by crosses, inhibitory areas by triangles.) In contrast, some of the cells, like simple cortex, have a side-to-side arrangement of excitatory and inhibitory areas with separation of the areas by parallel straight-line boundaries rather
Figure 4.4: The testing results of a 3-pixel wide bar strips image.

(a). Raw image: 3-pixel wide bar strips.

(b). Neural network result after thinning.

(c). Canny's detecting result: $\sigma = 0.7$.

(d). Canny's detecting result: $\sigma = 1.5$.

Figure 4.4: The testing results of a 3-pixel wide bar strips image.
(a). Raw image: 2-pixel wide bar strips.

(b). Neural network result after thinning.

(c). Canny's detecting result: $\sigma = 0.7$.

(d). Canny's detecting result: $\sigma = 1.5$.

Figure 4.5: The testing results of a 2-pixel wide bar strips image.
than circular ones. There are several varieties of fields, differing in the number of subdivisions and the relative area occupied by each subdivision. The commonest arrangements are illustrated in figure (4.6 (c) (d) (e) (f) (g)). The orientation is a characteristic of each cortical cell, and may be vertical, horizontal or oblique. There is no indication that any one orientation is more common than the others. For maximum center response the orientation of the slit is critical; changing the orientation by a little bit (such as $5^\circ - 10^\circ$) is enough to reduce a response greatly or even abolish it.

Figure 4.6: Common arrangements of lateral geniculate and cortical receptive fields. (a) 'on' center geniculate receptive field. (b) 'off' center geniculate receptive field. (c) - (g) Various arrangements of simple cortical receptive fields. $\times$, areas giving excitatory responses; $\Delta$, areas giving inhibitory responses. (Reprinted from [15]).

An appropriate description of these receptive field profiles as oriented Gabor filter or derivative-of-Gaussian filters was proposed in 1980. These filters are two-dimensional bandpass filters, which respond only over a limited range of orientations and of spatial frequencies. An example is given in figure (4.7). Three raw 2-D receptive fields profiles measured in cat visual cortex by L. Palmer and J. Jones are shown in (a), the best-fitting members of the family of 2-D Gabor filters is shown in (b) (so named because the functional form generalizes the 1-D elementary signals proposed in Gabor's (1946) famous Theory of communication), and the difference is shown in (c). In 97% of the
130 simple cells studied, these residuals of the fits are statistically indistinguishable from random error, confirming the appropriateness of this family of elementary filters for describing the neural primitives employed in low-level biological vision.

Figure 4.7: (a) 2-D receptive fields profiles of simple cells in cat visual cortex, recorded by L. Palmer and J. Jones. (b) best-fitting 2-D Gabor filters. (c) residual error. (Reprinted from [33]).

4.2.2 Neural Network Weighting Patterns

Before training, the weighting patterns are set quite random. After prolonged training, the weights are converged to regular patterns. The hidden units in the most common form of the back-propagation algorithm respond to a linear weighted sum of their inputs, they can be viewed as forming linear spatial filters through the modification of their input weights.

The weights to and from some of the hidden units are shown below. Each edge is detected by activity in many hidden units and each hidden unit contributes to the detection of many edges, in particular, several hidden units can cooperate together to detect some very difficult cases, and also they are able to discriminate formant-like image
segment in the presence of noise. It is very interesting to investigate why some filters are the way they are.

The weighting patterns shown in figure (4.8) are similar to non-oriented Gaussian filter. They have a strong positive center and a circularly symmetric inhibitory surround, or a strong negative center and a circular symmetric excitatory surround, which just resemble the “on-center-off-surround” or “off-center-on-surround” receptive fields isotropic bandpass spatial frequency filters found for retinal ganglion cells of the cat and monkey [15].

Figure 4.8: (a). The isotropic weighting patterns, which are similar to non-oriented Gaussian filters. (b). The 3-D plot of the isotropic weighting patterns.

These isotropic bandpass spatial frequency filters can have the functionality of smoothing and blurring, filtering out high frequency noise, to make the image easier to detect for the network.

Some of the weighting patterns shown in figure (4.9 – 4.16) are similar to oriented Gaussian derivative filters, with each pattern responsible for a very narrow limited range of frequencies. The orientations of edges are very critical, different oriented filter deals with different orientation, only those edges whose orientations fall in this range can have quite strong response. Usually, parallel columns can be easily noticed from the weighting patterns, but the displacement can be very different.

Even with the same orientation selectivity, different filters are responsible for different cases. Some are tuned to long and thin edges, but some are for short bars.

As we divide the plan into eight parts, and we train the network to learn eight oriented visual primitives, so we can see these oriented filters can cover these eights oriented parts. When an image segment is applied to the network, the hidden units which has the similar arrangement are most likely to be fired.
Figure 4.9: (a). The oriented weighting patterns for edges in the range of $[0^\circ, 22.5^\circ]$. (b). Some of the 3-D plot of the oriented weighting patterns.

Figure 4.10: (a). The oriented weighting patterns for edges in the range of $[22.5^\circ, 45^\circ]$. (b). The of the 3-D plot of the oriented weighting patterns.

Figure 4.11: (a). The oriented weighting patterns for edges in the range of $[45^\circ, 67.5^\circ]$. (b). Some of the 3-D plot of the oriented weighting patterns.
Figure 4.12: (a). The oriented weighting patterns for edges in the range of $[67.5^\circ, 90^\circ]$. (b). The of the 3-D plot of the oriented weighting patterns.

Figure 4.13: (a). The oriented weighting patterns for edges in the range of $[90^\circ, 112.5^\circ]$. (b). The of the 3-D plot of the oriented weighting patterns.

Figure 4.14: (a). The oriented weighting patterns for edges in the range of $[112.5^\circ, 135^\circ]$. (b). The of the 3-D plot of the oriented weighting patterns.
Figure 4.15: (a). The oriented weighting patterns for edges in the range of $[135^\circ, 157.5^\circ]$. (b). The of the 3-D plot of the oriented weighting patterns.

Figure 4.16: (a). The oriented weighting patterns for edges in the range of $[157.5^\circ, 180^\circ]$. (b). The of the 3-D plot of the oriented weighting patterns.
The large-scale oriented filters are tuned to relatively wide range of frequencies, which can effectively detect quite blurred image, see figure (4.17).

Figure 4.17: (a). The large-scale oriented filters. (b). The 3-D plot of the oriented weighting patterns.

There are some filters which have regular patterns but the functionality is not very clear, it can be filters for corners or some other complicated visual primitives, or it plays the assistant roles to cooperate with other neurons to fulfill the complex tasks.
Chapter 5

Discussion

In this last chapter, we conclude the thesis with a summary and problems of the neural network approach for edge detection. Future work and some possible extensions are discussed in the last section.

5.1 Summary

This work is inspired by the early processing in visual cortex of biological systems. Orientation of edges and contour elements of an image is considered to be one of the main features detected by neurons of the primary visual cortex. Orientation-selective properties of these neurons make it possible to encode orientations of image elements and to compress initial information.

As edge detection is the foundation of high level vision, the aim of the work is to develop a set of linear filters by using an artificial neural network to fulfill the task of edge detection. Currently, lots of edge detectors have been designed and implemented, but how to effectively deal with multiple scale intensity changes is still a hard problem for vision researchers. In addition, these edge detectors do not take account of orientations in the same way as the visual cortex of biological systems. The neural network approach used in this thesis is conceived to overcome these problems and to simulate the visual cortex processing to get a multiple scaled and oriented edge detector.

A three-layered feed-forward neural network architecture is built for this model, and the back-propagation algorithm is used to keep modifying the weights connecting layers. The hidden units of the network take the linear weighted sum of their inputs and form a set of linear filters.

As the back-propagation learning algorithm is a kind of supervised learning, the
major part of this work is to generate the appropriate input-output pairs. Various kinds of training images are designed for the use of input of the network, such as the images with sharp edges and blurred edges, low contrast and high contrast images, images with overlapped polygons, images heavily polluted by noise, images with short bars, images with long and thin lines, images with small blobs, etc. Each time a $7 \times 7$ square window is taken from the novel generated image and fed to the input layer (which contains 49 receptors). The center pixel of this small square is checked by the output layer. 17 visual primitives are learned by the network, with 1 general edge marker and 16 directional detectors.

During the training period, the weights connected the hidden layer with the output layer and the weights connected the input layer with the hidden layer are modified by the learning algorithm. Finally, the weights are converged to a stable state, and the weight patterns become regular.

After analyzing the hidden units, we find that the weight patterns connected to the hidden layer bear a resemblance to the receptive fields of simple cells in primary visual cortex, which is close to the Gabor and the derivative of Gaussian filters, including the non-oriented (such as Laplacian of Gaussian) and oriented (high order derivative of Gaussian) filters.

The testing results are also further processed by thinning and subpixel calculating algorithm.

By comparing with other existing edge detectors, we find the performance of the network as an edge detector provides very significant detail and appears to be superior in some aspects.

5.2 Problems

Although the edge detector developed by neural network has some advantages, it still has some drawbacks and problems.

During the training period, the back-propagation algorithm with rescaling variables is used, which is a small improvement of standard back-propagation algorithm, adding a multiplier to the back-propagated error when modifying the weights. Actually, the learning is still very slow; it needs a long time to do an experiment. So, it costs lots of computer resources – time to train the network to be stable and the space to store many training patterns.
The second problem is shown in the neural network testing results. This edge detector is very good at dealing with the intensities above some value, and responds not so well when the intensity is too low. This drawback is also due to the training algorithm. If we take a look at the formula of changing weights in the general delta rule, we can find that when the output of an neuron is too small, only a little modification is made. In addition, to detect the low contrast edges is still not as easy as to detect the high contrast edges for the network.

There are also a few problems in the thinning algorithm and subpixel calculation. We keep tracing an edge in one direction until its response is too weak, then change to another direction. This method is good for long and straight lines; but for small features, like blobs, it can make the direction change too sharp, so the small curve is not so smooth. Also, the non-maximum suppression used in our method sometimes makes the T-junctions or V-junctions disconnected. So, we need further smoothing and linking process.

5.3 Future Improvements

There are several possible and interesting extensions.

One of the obvious improvements is to add the visual primitive of junction [21]. "V" junctions, "T" junctions and "Y" junctions can be considered. A "V" junction is formed by two edges terminating at the same point, a "T" junction means one edge terminating at another, whereas a "Y" junction means three edges terminating at one point. The value of the corner output unit can be designed as the inverse of the distance from the center pixel to the intersection point of multiple edges (at least two edges). The maximum value can be set to 0.9 which means the center pixel is just falling on the intersection; while the minimal value 0.1 means it is too far from the intersection. For the neural network architecture, we suggest that either three output units be added to the output layer of the existing network to represent the corner information, or one more layer with three output units be put on the top of the existed network. Figure (5.1) shows three possible architectures to detect the corner information.

The corner information is very important for edge detection. With the corner value, the detecting results can be much improved.

Another similar extension is to add a curvature visual primitive. The training input images can be designed as circles with different radius $R$, the corresponding desired
Figure 5.1: Three possible architectures for detecting corner information.
output is \( k/R \), where \( k \) is a constant to make the desired output value in the range of \([0.1, 0.9]\). Again, the suggested architecture is three-layered or four-layered neural network, depending on whether the contour curvature is treated as the basic visual primitive or it is built on other visual primitives [21]. With this visual primitive, we can measure the curvature of any contour so that we can trace the contour, including some blobs, very well.

In addition to the two visual primitives mentioned above, we can measure the blurring of an edge by using more output units. The input image can be sharp images or blurred images. For the sharp images, the desired output is 0.1 to denote no blurring at all. Several blurring mask can be designed to get different levels of blurring. With blurring information, we can distinguish sharp edges from blurred edges, even more, we can detect how much one edge is blurred.

Finally, one of the most attractive directions of improvement is to detect subjective contours [1], see figure (5.2).

![Figure 5.2: Subjective contours. (Reprinted from [1]).](image)

It is not very clear how to generate the input and desired output pairs, and what kind of neural network architecture is appropriate for this problem, but it is a very interesting direction to pursue.

Over all, by performing systematic experiments with different desired outputs, it should be possible to determine the function of various categories of neurons in early vision for biological systems.
Bibliography


