

Evolving ANN for Edge Detection

Craig C. Ewert
DePaul University
Chicago, IL

cewert@shrike.depaul.edu

University of Maryland University College
RAF Menwith Hill
cewert@faculty.ed.umuc.edu

Yakov Keselman
DePaul University
Chicago, IL
ykeselman@cti.depaul.edu

Abstract

The authors investigate the use of a genetic algorithm to control the evolution of artificial neural networks for the purpose of detecting edges in single-line digitized images. Special-purpose neurons are used in an architecture which is fixed in terms of inter-layer connectivity but free in terms of number of layers, activation functions, and other attributes. Preliminary results indicate that three of the free attributes do have a best form.

I. Introduction

The field of computer vision uses edge detection as part of the process of recognizing objects in a digitized image. Once the edges of objects are detected in the image, other techniques can be used in an attempt to identify what particular objects are in the scene and where they are located. Several algorithmic methods have been developed for edge detection, with LaPlace, Sobel, and Canny being most common [6].

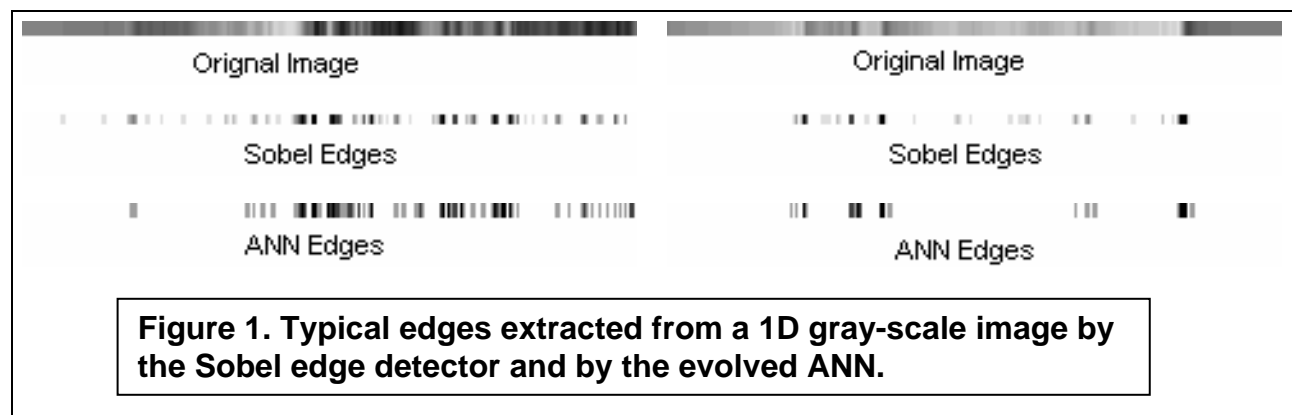
The question that we tried to answer in this research project is whether it is possible to realize an edge detector as an artificial neural network (hereafter referred to as an ANN) [2] and how one can arrive at the desired architecture. For training purposes, we took the Sobel edge detector. Thus, the question being addressed in the paper is how one can arrive at an ANN architecture that will simulate the Sobel edge detector. Even though ANNs are notoriously difficult to train and to “understand”, they can be an attractive alternative to the current edge detectors, as they can provide results with relatively little computational effort.

Most ANN training approaches assume the ANN architecture to be fixed, and adjust edge weights according to a specified procedure. Even then, traditional training methods for ANNs suffer from localized search, frequently getting trapped in local optima. If the architecture itself is allowed to change (evolve), the size of the search space increases dramatically, which renders traditional training methods practically useless. On the other hand, genetic algorithms (hereafter

referred to as GA) [1] provide a method of investigating a large area of the search space in parallel. In addition, our desire to consider ANN architectures that use step-wise perceptrons (which means that the error surface of the ANN is not differentiable) precludes us from using an existing training method for ANNs with this activation function.

These considerations led us to the problem of trying to arrive at an ANN architecture that will simulate the Sobel edge detector using GAs to explore the search space. To simplify the problem, we used gray-scale one-dimensional “images”, which were obtained by extracting a single row from an existing two-dimensional gray-scale image. The row was duplicated several times to create a two-dimensional image to which the Sobel edge detector can be applied.

Two examples of such images and edge pixels extracted from them are shown below. In those examples, the darker a pixel is, the more edge-like it is. Please note that even though our evolved ANN may sometimes give “perceptually better” results than the Sobel edge detector, our overall objective is to simulate that edge detector as closely as possible. The reader can judge for him/herself how well the objective was achieved based on the examples in Figure 1. A summary of results is given in a later section.



II. Approach and Design

The overall approach can be summarized as follows. Starting with a “random” ANN that has as many neurons in the input layers as there are pixels in the input image, evolve its architecture, changing various characteristics of the ANN as deemed necessary. To drive the evolution process, use the GA technique, as described below. In what follows, we first review some concepts related to ANNs and GAs and then describe our scheme of using them together.

ANNs [2] have been extensively used to solve a variety of pattern-recognition problems. The most common approach is to define a network of artificial neurons, where each neuron takes some input and calculates an output value. The first layer takes input from the raw input values of the data, while the second layer takes input from the outputs of the first layer neurons. The output of the final layer is the solution derived from the input. Normally, the first layer neurons have a single input value – the data – while all neurons in a later layer take inputs from some or all neurons of the prior layer.

For our project, we decided to use a modified neuron rather than the standard ANN neuron. Based on the fact that edges are usually detected using differences in pixel intensities, instead of inputting a single pixel value to each input neuron, we took the absolute value of the difference

between two pixels as the input. We also decided to keep the same architecture for every layer which might be evolved in the ANN. Thus, for a second layer, each neuron would input the absolute difference of the output values of two neurons in the previous layer rather than a weighted sum of the outputs of some or all neurons in the previous layer.

We now summarize the GA approach [1] to solving optimization problems. In brief overview, a population of genomes – each of which represents a guess at a solution to the problem – is randomly generated. Then the performance of the solution represented by the genome is measured, to yield a fitness value. This fitness value is assigned to the genome that represents the solution. After all the genomes have had their fitness values derived, those values are scaled to yield a set of values which are of greater use in driving the natural selection process of evolution. A randomizing technique is then used to determine which genomes will be allowed to partake of parenting the next generation of genomes. This is normally done via recombination of portions of 2 different genomes, mimicking the sexual reproduction found so often in nature. The resulting offspring are then subjected to potential mutations – random changes in their genomic contents, most often a bit-flip $0 \leftrightarrow 1$. All members of the first generation then die, and are replaced by their offspring. The entire process is then repeated for the new generation, leading to the creation of yet another generation of genomes. This process can theoretically continue indefinitely, so some external stopping criterion is normally imposed to halt it.

When using GA to evolve solutions to problems, the major difficulties are (1) the genome representation of a solution, and (2) the particular objective function used to evaluate how well a particular solution fits the problem. We set our genome template to allow for several aspects of the solution to be evolved:

1. The activation function (4 different functions were available, and a choice was evolved. The possible choices were sigmoid, Gaussian, identity, and multi-level perceptron¹).
2. The number of layers in the ANN (which could vary from 1 to 7).
3. The threshold and firing levels for the perceptrons (from 0 to 255).
4. The number of levels for the perceptrons (initially 3, but could grow – largest reached 63).
5. The separation of input pixels or neurons for the next layer of the ANN, used to determine input values. Also referred to as skip count (e.g., adjacent pixels have skip count of 0).
6. The amplification factor for neuron output (positive integer, used when the activation function was sigmoid or Gaussian, to develop an output greater than unity).

The objective function we chose was the inverse of the sum of the absolute value of the differences between the Sobel-detected pixel value and the ANN-predicted pixel value. This maintained the normal GA relationship that a higher fitness value denotes a better solution, since the sum increases with poorer performance.

Our GA technique used a fitness-proportionate mating selection scheme, a population size fixed at 100 individuals, a mutation rate of 0.003/bit, and a run of 20000 generations at a time. The fitness values of individual genomes were then scaled via the formula $F = F - F_{avg} + (2.5 * \text{standard deviation})$, as recommended in [1], to guard against premature convergence in the early stages and to maintain the natural selection pressures during the later stages. Chances of recombining during mating were set at 0.7, thus ensuring that some genomes would survive unchanged into the next generation. No elitist methods were used, so the surviving genomes were not guaranteed to be the best of the prior generation, but a randomly diverse subpopulation.

¹ A perceptron has a single threshold. If the input value exceeds the threshold, the neuron fires and the output is 1, else 0. Our multi-level perceptrons have multiple thresholds, and each threshold has its own output value. The threshold values and output values both evolve during the GA runs, along with the number of thresholds.

III. Results and Observations

Ten runs of 20000 generations each were made, and the best-performing run was then extended to 160000 generations. Results from these runs are tabulated elsewhere². The best ANN performance had pixel values which, on average, were 15.76 away from the Sobel pixels, with an objective function value of just over 4.4E-5.

There is no dramatic evidence that any particular number of layers in the ANN is significant, as similar performance was realized with 1, 2, 3, 4, and 6 layers. The lack of 7 layer ANNs, and the single layer 5 ANN, may easily be attributed to statistical variance in so small a number of runs.

For the skip count, the evidence is rather dramatic that 1 pixel should be skipped. Of our 10 runs, there were 1175 layers using a skip count of 1. The nearest competitor is a skip count of 5, with 529 layers using this – less than half as many. This was rather surprising, as we had anticipated a skip count of 0 as being most likely – because an immediate significant change in pixel values would indicate a sharp edge.

In terms of the activation function, there are 1200 layers in our initial 10 runs using the multi-level perceptron, with 662 (slightly over half) using the next most common – the identity function. This strongly suggests that the multi-level perceptron is the preferred activation function for this activity. The relative constancy of the perceptron layers in providing good results seems quite striking in later generations.

IV. Future Directions

This research has shown that three of the attributes we were investigating – the activation function, the number of layers in the ANN, and the separation of pixels/neurons for inputs – have definite values which are most appropriate for this work. Using this knowledge, the search space for solutions is reduced considerably.

We plan to alter the GA and genomes used accordingly, and continue with this research. The objective function will be modified to encourage smaller genomes, which will use only multi-level perceptrons with sorted thresholds (thresholds were unsorted in this research). ANN layers will be limited to a maximum of 2. The skip count will be allowed to vary from 0 to 1.

We would also like to assess the impact of the mutation rate on the resulting performance. With the currently used mutation rate of 0.003, the probability of a mutation is slightly over 0.5 per 30 bytes, which makes a mutation almost inevitable with the genome size starting at 126 bytes. Smaller mutation rates will perhaps give us qualitatively different results.

V. Related Work

The use of GAs to control the evolution of ANNs is not new to this research. In particular, [7], [10], [11], [12], and [13] all do exactly that. [8] and [9] use GAs by themselves in an attempt to solve particular problems in computer vision, while [10], [11] and [12] use GAs to evolve ANNs for a different purpose than computer vision. [15] provides a good overview of GA, and the historical development of the field. [14] provides an excellent overview of the extent of work being done in connection with GA control of ANN evolution, listing several

² At <http://shrike.depaul.edu/~cewert/fall2003.xls>

hundred papers. This is a rich field for further research, as it is quite young and unfinished at the current time. [16] provides a good tutorial on the use of Sobel and LaPlace algorithms for edge detection, including the implementation of the algorithms in C code.

V. References:

1. Goldberg, David E., *Genetic Algorithms in Search, Optimization & Machine Learning*, 1989, Addison-Wesley
2. Fausett, Laurene, *Fundamentals of Neural Networks Architectures, Algorithms, and Applications*, 1994, Prentice-Hall
3. Mitchell, Tom M., *Machine Learning*, 1997, McGraw-Hill
4. Banzhaf, Wolfgang, et. al., *Genetic Programming An Introduction*, 1998, Morgan Kaufmann
5. Hoffman, Donald D., *Visual Intelligence How We Create What We See*, 1998, W. W. Norton
6. Sonka, Milan, Vaclav Hlavac, and Roger Boyle, *Image Processing, Analysis, and Machine Vision*, 1999, Brooks/Cole Publishing Company
7. Huang, J.-S. and H.-C. Liu, *Object Recognition Using Genetic Algorithms with a Hopfield's Neural Model*, 1997, *Expert Systems With Applications* 13(3): 191-199.
8. Singh, M., A. Chatterjee, et al., *Matching Structural Shape Descriptions Using Genetic Algorithms*, 1997, *Pattern Recognition* 30(9):1451-1462
9. Yuen, S. Y. and C. H. Ma, *Genetic algorithm with competitive image labelling and least square*, 2000, *Pattern Recognition* 33: 1949-1966
10. Hamada, H. and M. J. Reyes, *User-defined Object Classifier based in a Neural Network with Optimal Selection of Visual Features using a Genetic Algorithm*, 1999, School of Engineering and Applied Science, Columbia University, New York, NY., at <http://www.ee.columbia.edu/~mjr59/final1.pdf>
11. Moriarty, D. E. and R. Miikkulainen, *Forming Neural Networks through Efficient and Adaptive Coevolution*, 1998, *Evolutionary Computation* 5(4)
12. Stanley, K. O. and R. Miikkulainen, *Continual Coevolution through Complexification*, 2002, University of Texas at Austin, at http://www.cs.utexas.edu/users/nn/downloads/papers/stanley.gecco02_2.pdf
13. Yong, C. H. and R. Miikkulainen, *Cooperative coevolution of Multi-Agent Systems*, 2001, University of Texas at Austin, at <http://www.cs.utexas.edu/users/nn/downloads/papers/yong.tr287.pdf>
14. Alendar, Jarmo T., *Indexed Bibliography of Genetic Algorithms and Neural Networks*, 1995, at <ftp://ftp.uwasa.fi/cs/report94-1/gaNNbib.ps.Z>.
15. Heitkotter, Jorg and Beasley, David, *The Hitch-Hiker's Guide to Evolutionary Computation (FAQ for comp.ai.genetic) Issue 8.2*, 2000, at <http://surf.de.uu.net/encore/www/>
16. Green, William, *Edge Detection Tutorial*, 2002, at <http://www.pages.drexel.edu/~weg22/edge.html>