Evolving facial recognition

Sean Dawson

November 11, 2011

1 Abstract

Using back-propagation to train artificial neural networks (ANNs) has been popular since the seventies. Although it is a thoroughly tried and tested algorithm, it is now forty years old. This study attempts to apply the more modern algorithm of Particle Swarm Optimisation to training ANNs to determine whether back propagation is showing its age. To compare the algorithms, an ANN is trained to recognise the presence of a face in a when presented with an image.

Contents

2 Introduction

Facial recognition is a task that traditional computing methods persistently have trouble with. The many variables involved such as differing light conditions, facial sizes and positions, and variations in complexion are just a few of the factors that make it a difficult process for the traditional computer to handle. Humans excel at facial recognition due to the flexibility of the neural networks that form their brains. From this we can infer that an artificial neural network (ANN) may perform with a similar aptitude given that it is trained optimally. That said, training an ANN is a difficult task. The weights of an complex ANN cannot be determined by hard coding and deriving a correct set of weights presents an extremely large search space. This presents a perfect opportunity to utilise evolutionary computing concepts.
A common way to train networks is via back propagation, an idea that has in use since around 1969 [9]. It involves determining the error of the outputs and working backwards towards the inputs adjusting the weights incrementally to reduce the error, commonly using a method such as stochastic gradient decent. It has been criticized in some cases for being slow, and prone to being stuck in local minima [1,3,11]. This study proposed that by using the more modern technique of particle swarm optimisation (PSO) an ANN could be trained in less time and result in a set of weights that result in a more adaptable network, especially in larger networks. Experiments were performed that investigated the efficiency of both algorithms, and also gauged how effective the final weighting configuration is at detecting faces that were not in its training set, referred to as it's adaptability. It is predicted that when optimising an ANN for a complex task such as facial recognition, PSO will excel due to its explorative nature and its diverse population, trailing many more candidate solutions than back propagation. Although there has been some research into training neural networks with PSO [13,14], there was a gap in research directly comparing it to back propagation with its effectiveness in optimising facial recognition networks. This is what this study planned to address. PSO was chosen out of the several evolutionary optimisation techniques due to its excellent performance in a vast array of optimisation problems.

3 Literature Review

'A Look at Facial Recognition' by J. Woodward et al [2] presents the idea of using facial recognition as a biometric which is a way to distinguish one person from another automatically. The paper investigates the use of facial recognition in Virginia Beach, US to identify criminals and missing persons using CCTV cameras. The facial recognition software alerts the police headquarters with possible matches where the footage is reviewed by a trained officer to determine false alarms. It discusses facial recognition in a more general overview, rather than the exact implementation of the software and gives good examples of potential pitfalls and methods used in facial detection in busy environments. It introduces concepts such as using correct training data which achieves the balance between difficulty and simplicity for the recognition software so that it isn't over trained to either ends of the scale.

'Neural Network-Based Face Detection' by H. Rowley et al [8] provides a more in-depth look at the mechanics of facial recognition and the part ANNs can play. The method utilized by this article uses a 20x20 pixel grid for the input to the ANN. This grid scans over the top of the image detecting for the presence of a face. The source image that is being scanned is repetitively sub sampled or shrunk and then rescanned so that different sized faces can be detected. The layout of the ANN internal/hidden nodes are in 3 distinct groups that the paper calls 'Retinally connected'. One group of hidden nodes samples 4 10x10 sub regions of the input, another group samples 20 5x5 sub regions and yet another section samples the input in overlapping horizontal stripes. The overlapping stripes were included to try and pick up horizontal features such as a pair of eyes, the brow or even the mouth. The training method used in this paper is standard back propagation with momentum. This article also gives a good overview of possible pitfalls and countless methods that could be implemented into this research.

'Evolving Artificial Neural Networks' by Xin Yao [12] is a excellent article that compares training an ANN via evolutionary algorithms with traditional methods. It introduces an interesting topic that the actual architecture of the ANN could be evolved rather than just strictly the weights. The overall conclusion of this article suggests that both evolutionary methods and traditional methods are effective depending on the problem presented. It does mention that evolutionary algorithms excel at higher complexity levels which supports my proposal, but it presents no hard evidence of the superiority of evolutionary methods in ANN training.

4 The Model

4.1 The Artificial Neural Network

An Artificial Neural Network (ANN) is a network of interconnected nodes with weighted connections. An ANN can take the form of many architectures and designs. In the scope of this project, a 3 layer feed forward network is used, also known an a perceptron. A perceptron consists of an input layer, a hidden layer and an output layer. Each node sums the weighted outputs from the layer above it, applies a bias and then outputs to the next layer. A perceptron is one of the most simple architectures of neural networks but is still a very powerful classification tool. The weighting on each connection has to be tuned so that the neural network can become useful. This is never done manually in all but the simplest of networks therefore, in this project, an algorithm is used to optimise the weights. An example of a widely used training algorithm is Back-propagation. (See 4.2.1)

The images that are presented to the neural network for recognition are split up into segments and each segment is connected to a unique hidden node. This was based on the work of Rowley et al [8] in which they likened this method to a human retina. Due to time constraints the representation used in these experiments was set to a fixed grid. Measuring the effects of different representations of the images could be the subject of further research but is beyond the scope of this project.

Initialization In the network that is to be optimised, initially each weight is set to a random distribution between $-n$ and n where n called the Nguyen Widrow factor. The Nguyen Widrow factor is calculated as follows:

$$n = 0.7 * H^{\frac{1}{i}}; \tag{1}$$

Where:

H = Number of hidden nodes.

i = Number of input nodes.

Initializing neural networks with numbers in this range is known to speed up optimisation times with a broad range of applications [4].

4.2 The Algorithms

The two algorithms that are being compared both have the same objective; to adjust a set of weights in an ANN to minimize the mean squared error. Although the goals are the same, the solution is found by using very distinct methods.

4.2.1 Back-propagation

Back-propagation is an extremely common way to train standard feed-forward ANNs and is a type of stochastic gradient decent algorithm. It works by taking a sample from the training set at random and calculating the error at each synapse by comparing the expected output with the actual output. It then goes through every weight and adjusts them to minimize error. When every sample has been tested the epoch (or iteration) is complete. This is repeated many times until the mean squared error falls below an acceptable error. The formula that is applied to optimise each weight on a connection is defined as follows:

$$\Delta w = (l * o_{source} * e_{target}) + (m * \Delta w_{prev}) \tag{2}$$

Where:

l = Learning rate.

o_{source} = The current output of the source neuron

e_{target} = The error of the target neuron (Output - Expected Output).

m = Momentum rate.

Δw_{prev} = Previous change in weight.

Momentum Momentum is a feature which can help achieve faster convergence and also prevent noise in the training data from negatively impacting the weight calculations. It is implemented by keeping track of previous changes to a weight in the training process and adding it onto the current iteration smoothing out erratic changes in direction. The momentum rate is hard coded to 0.07 in the NeuronDotNet library.

Learning rate The learning rate adjusts the decent rate of the gradient. If it is set to a number that is too small it can get stuck in local minima and if set too large then the weights can diverge, the solution can be overshot and there can be lot of oscillation in the error function. A learning rate of 0.3 is a good place to start and is the default in NeuronDotNet.

The learning rate can be set as a function so that it decreases with time. This can be used to speed up the initial process by starting with a large learning rate and then reducing it down as the network learns to become more accurate. In NeuronDotNet there are linear, exponential and hyperbolic functions to adjust the learning rate dynamically.

Jitter Jitter is a feature provided by the NeuronDotNet library that tries to prevent back propagation from getting stuck in local minima. It applies a small amount of noise to the connection weights every time the jitter epoch occurs (by default every 73 iterations).

4.2.2 Particle Swarm Optimisation

Particle Swarm Optimisation was devised originally to simulate social behaviors but it suited the task of optimisation well and ever since it's inception it has become a major player in evolutionary computing. It works on using a 'swarm' of candidate solutions referred to as particles and the attraction between them to search for optimum solutions. Each particle in the swarm has a position in n-dimensional space which represents a candidate solution. Each particle also has a velocity which modifies its position. The main equation in the original non problem specific PSO that governs particle velocity on each generation is defined as:

$$\begin{cases} \vec{v}_i \leftarrow \chi(\vec{v}_i + \vec{U}(0,\phi_1) \otimes (\vec{p}_i - \vec{x}_i)) + \vec{U}(0,\phi_2) \otimes (\vec{p}_g - \vec{x}_i)), \\ \vec{x}_i \leftarrow \vec{x}_i + \vec{v}_i, \end{cases} \tag{3}$$

Where:

$\vec{U}(0,\phi_i)$ = Is a random number between 0 and ϕ_i.

v = The velocity of the particle.

x = The current position of the particle.

$\vec{p}_i$ = The personal best of the particle.

$\vec{p}_g$ = The global best position.

χ = Constriction factor.

In other words, on each iteration the velocity of each particle is modified by a random factor but is also influenced by its personal best and the global best. χ is known as the constriction factor and was suggested as a replacement to a clamped maximum velocity to prevent the particles becoming too excited. ϕ_i is another constriction factor and represents the range of the random distribution that is applied to the velocity.

Standard PSO There are many implementations of PSO but this project used Standard PSO, a variation that was developed as a baseline to compare other implementations with. As this project is about comparing PSO to back-propagation rather then comparing different PSO implementations, it is fitting to use the baseline so that it could be improved upon in later experiments.

Instead of modifying each particle's velocity on the global best, Standard PSO uses what is known as a LBest or local best topology. In the LBest topology, particles form small local neighborhoods that influence each other. The amount of particles that are in each neighborhood is governed by the variable k which is set to 3 by default. The neighborhoods are not static and reinitialize at certain times. An LBest topology is known to converge slower, but it is more thorough in its search and the particles can focus on several potential solutions before finally converging into one.

There is a limit to the position of each particle: if one dimension exceeds the maximum or minimum limits of a run then it is clamped back to the limit and its velocity is set to 0.

The version of Standard PSO used in this project was ported to C# by Travis Silvers but the functionality is identical.

The parameters used in Standard PSO and referred to in this article are:

Default Parameters	
iMin	Minimum start position of a particle
iMax	Maximum start position of a particle
pMin	Minimum position of a particle
pMax	Maximum position of a particle
d	Number of dimensions
s	Swarm Size
k	Maximum neighborhood size
p	Probability of a one particle informing another
w	First constriction factor (χ)
c	Second constriction factor (ϕ)

For every experiment, maximum and minimum start positions were set to the Nguyen Widrow factor. (See 4.1)

Default Parameters The default parameters used for the PSO (as taken from Standard PSO) were:

Default Parameters	
s	$10 + 2\sqrt{d}$
k	3
p	$1 - (1 - \frac{1}{s})^k$
w	$\frac{1}{\log(2)}$
c	$0.5 + log(2)$

4.3 Research Software

To run the tests and process the results a software package called E.N.F.O.R.M.[1] was created. The software package consists of 6 main sections:

- 8.1.1 Run Editor
- 8.1.2 Optimiser
- 8.1.3 Network Tester
- 8.1.4 Results Analyser
- 8.1.6 Face Explorer
- 8.1.5 Image Tool

See the appendix for a detailed explanation on E.N.F.O.R.M.

4.4 Testing Methodology

4.4.1 Creating a test set

The basis of all the research in this project are the training sets used during the optimisation and the testing sets used to verify the results. It is therefore prudent that these sets are designed to a high standard before any experiment is performed.

The F.E.R.E.T. Database The source of all the facial images for this experiment were from the F.E.R.E.T. database held at the NIST website [6]. It contains over 800 facial images that are catalogued with text files describing them. The database has a great variety of faces to use in facial recognition and contains people of many different ethnicities, ages and genders. The first step was to collate all the text files that described each photo and put it into an SQLite database so that they could be organised properly. The resulting database was used by the face explorer tool (See 8.1.6) to allow the organisation of images into sets.

The Non-facial Images Most of the non-facial images were sourced from a website called 'Easy Stock Photos'. The photos are public domain and therefore able to be used in this experiment without paying royalties. There were approximately 800 images in this collection which coincided with the amount of faces in the F.E.R.E.T. database well. There were thirty five categories of images which gave a great variety to the set. There was a large diversity between each image in every category also which helped to avoid over training. There were also some other photos added such as game screen shots and photos of objects that look roughly like faces in an attempt to produce higher quality results. The full list of image categories is listed in the appendix on page 22.

[1]E.N.F.O.R.M.is an acronym representing Experimental Neural Facial recognition Optimisation Reporting and Management.

Organising the images into sets To create high quality training and test sets, the images need to be split in an effective way. Simply splitting the set of images in half would result in biases within each set. For example, the training set might consist of only men and the testing set of only women. This kind of situation was avoided in this project by separating the images into categories and splitting them. When splitting the face images, first the faces with glasses (as there were only a small number of them) were taken out and split. The same thing was done for faces with beards and mustaches. The remaining faces were split into male and female, and then each ethnic group was split into each set. When creating the set of non-facial images, each category was simply split in half.

Creating the baseline Before the experimentation could start a target needed to be specified otherwise the each experiment would run forever. For the purposes of this experiment it was decided that the ability to correctly recognise the images 100% was unachievable considering time constraints and the number of experiments that needed to be performed. A target of 99.5% success was determined. Converting this into a minimum error of 0.005, the target mean error squared was set to $0.005^2 = 0.000025$. To create a baseline to work off, both algorithms were run on default settings and the first one to reach the minimum mean squared error was taken as the benchmark.

Running experiments Once the benchmark was created, experiments were performed to try and get the other algorithm to perform at least as well as the benchmark. The actual experiments that were performed are outlined in the Result Analysis section. All experiments were averaged over ten runs.

5 Result Analysis

5.1 Determining the benchmark

For the first experiment, the images were scaled to 20x30 pixels with nearest-neighbor resizing. The image was sampled as 5x5 grid resulting in 25 hidden nodes. The total number of dimensions equated to 651. When the two algorithms were run on default settings, back-propagation out performed PSO by a large margin. After an average of 10 runs, back-propagation was able to reach the target MES in around 2 minutes and 23 seconds. Back-propagation was defined as the benchmark and the following experiments involved trying to improve the performance of PSO.

BP Parameters	
Learning rate	0.3
Jitter Epoch	73
Jitter Noise Limit	0.03

PSO Parameters			
pMin	-1	pMax	1
d	651	p	default
p	62	w	default
k	default	c	default

Figure 1: The parameters for the first experiment.

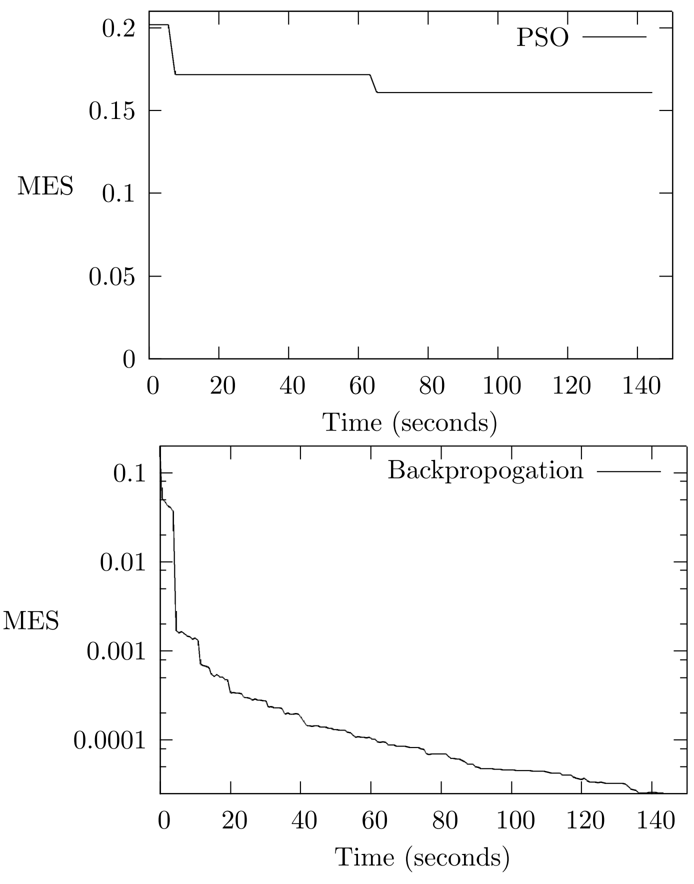


Figure 2: The initial runs. Note the scale differences between graphs.

5.2 Reducing image size

After the network was optimised by back-propagation it was tested on the training set to determine if it resulted in a high quality adaptable network. On average it recognised a face successfully 98% of the time and recognised a non-facial image 99.93% of the time. These are very good results and show that back-propagation produces high quality results as well as performing well.

It was believed that the large amount of dimensions may have been causing the PSO to perform poorly. The images were re sized to 18x12 pixels which was deemed the minimum to still have recognisable features. The edges of the image were also cropped to cut out the neck region and to remove some of the redundant white space around the edges of the faces. This decreased the amount of dimensions down to 289. This slowed down back-propagation to about 3 minutes and 3 seconds to reach the target MES. The performance of PSO reduced slightly.

Minimum fitness	
Back-propagation	0.000025
PSO	0.163258131

Figure 3: Minimum fitness after 3 minutes and 3 seconds.

5.3 Experimenting with parameters

After that, it was decided that the parameters for PSO would need to be changed from the default. Many different experiments were run with the results listed below.

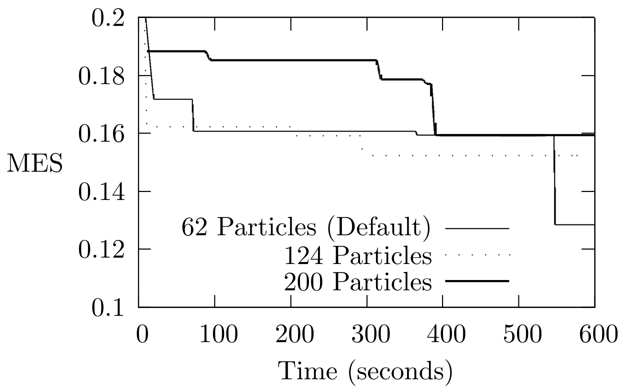


Figure 4: Experiments with different particle sizes.

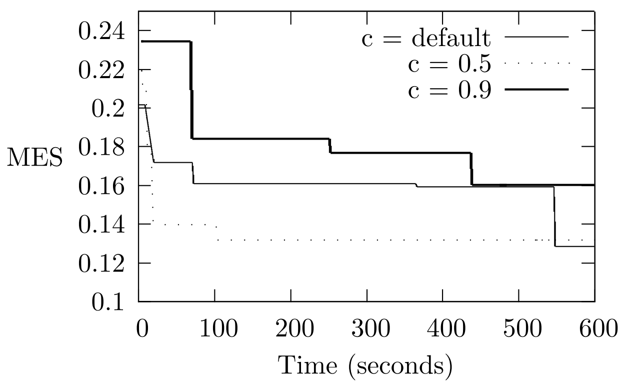


Figure 5: Experiments with different c values.

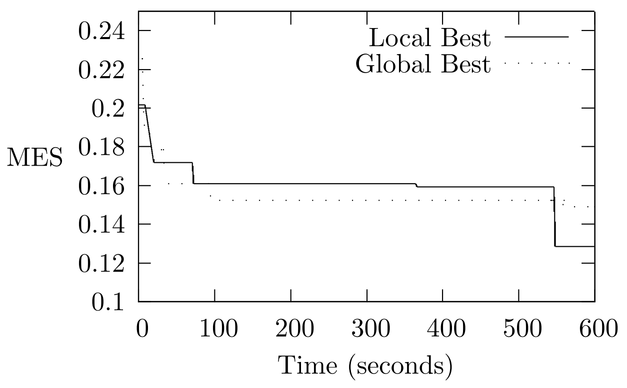


Figure 6: Experiments with LBest and GBest topologies.

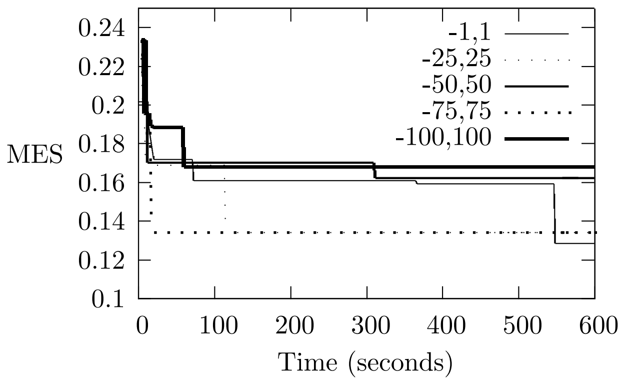


Figure 7: Experiments with different bounds.

As shown in these graphs, the results were very erratic. It appeared strange that the results improved with greater bounds on the particles. Considering that the search space of the problem is exclusively within the limits of neural network weighting (-1 to 1), the particles shouldn't be moving outside this range often. Also, the shape of the graphs seemed odd, with sudden jumps in fitness and then no improvement for long stretches of time. Code was written to determine how many particles were going out of bounds on each iteration and some runs were measured. Over the course of a standard PSO run with default parameters, it was found that on average about 1000 out of bounds were occurring every iteration. From this, the conclusion was drawn that the particles may be over excited and that they may need some dampening. Experiments were done by reducing the value of ω in increments of 0.1 until the best performance was found. It was found that the best value for ω is 0.2. The graph was also a much better shape, with constant improvement rather than erratic jumps in fitness but it still didn't come close to back-propagation in performance.

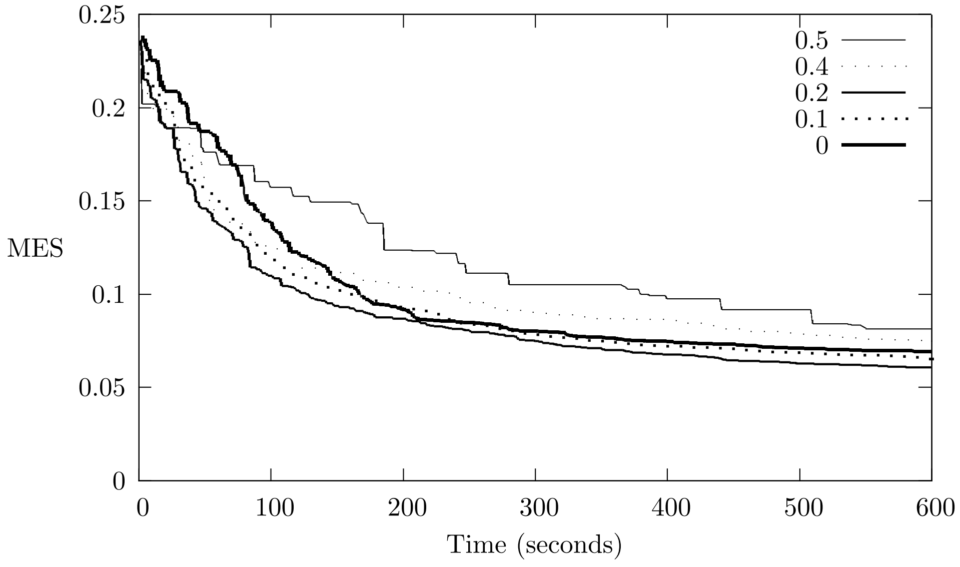


Figure 8: Experiments with different values of ω.

What is interesting is that even when ω was set to 0, the algorithm still functioned almost as well as when all the other values were used. With ω set to 0, random exploration is effectively disabled and the swarm moves only by the influence of its own personal best and the influence of its neighborhood.

5.4 Velocity Constraints

A paper by Xiaorong Pu et al [7] uses velocity constraints to try and reduce premature convergence, alleviate the effects of dimensional increases and prevent the particles from going out of bounds. The paper reported success with this method, so the PSO algorithm for this project was modified slightly.

On every iteration each particle was checked for velocities that exceeded a set maximum $vMax$. If a certain dimension of a particle was exceeding $vMax$ it was clamped to the average absolute velocity $v_{average}$ of every dimension of every particle.

$$v_{average} = \frac{1}{M * N} \sum_{i=1}^{M} \sum_{i=1}^{N} (|v_{ij}|) \tag{4}$$

Where:

M = Is the size of the swarm.

N = Is the number of dimensions.

i = Index of the particle.

j = The dimension of the particle.

v = Velocity.

After running experiments with the value of vMax from 0.2 to 1.4 in increments of 0.2, the best performance was found with 0.2. The values of 0.1 and 0.15 were then tested. The most interesting results are graphed in figure 9.

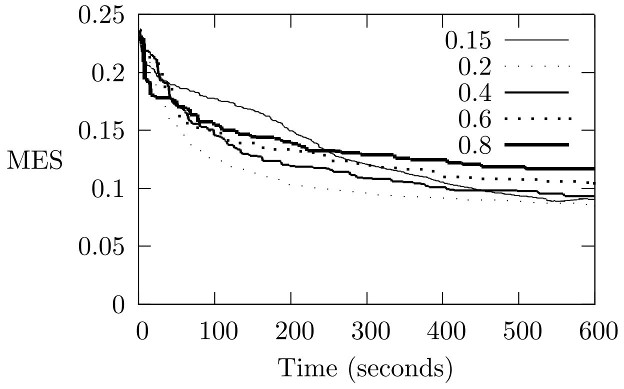


Figure 9: Experiments with different values of $vMax$.

This shows that the velocity clamping did improve the performance of PSO, but still not enough. The results are very similar to the best result in figure 8 when χ was set to 0.2. Next experiments were run that used velocity clamping with a χ value of 0.2.

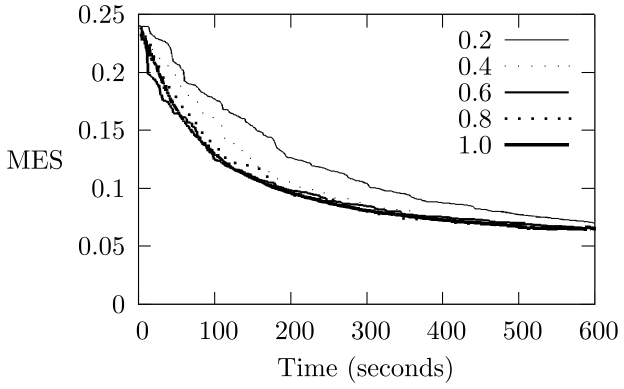


Figure 10: Experiments with different values of $vMax$ when $\chi = 0.2$.

It seems that combining the best value of χ found so far (0.2) with vMax clamping resulted in no net gain, as seen in figure 10. The results seemed to bunch together, most likely because the random factor was reduced.

6 Discussion

From these results, it seems that Particle Swarm Theory is not an ideal choice for training neural networks of this magnitude. Out of all the papers cited in this study, not one had anywhere near the amount of connections in their nueral networks that this projects did. Generally, experiments that train a neural network with PSO involve a fraction of the amount of nodes and connections that this experiment does [5,10,13]. Although there may still be a way to modify PSO to make it competitive with back-propagation, it was not found in these experiments.

It seems that when PSO encounters a problem with an excessive number of dimensions, the particle velocities can go out of control and need to be dampened so that the problem is explored correctly.

An interesting point of further research would be to change the PSO equation to use the error gradient at each connection to influence the particles. This would make the particle swarm more informed and would combine the best of both worlds.

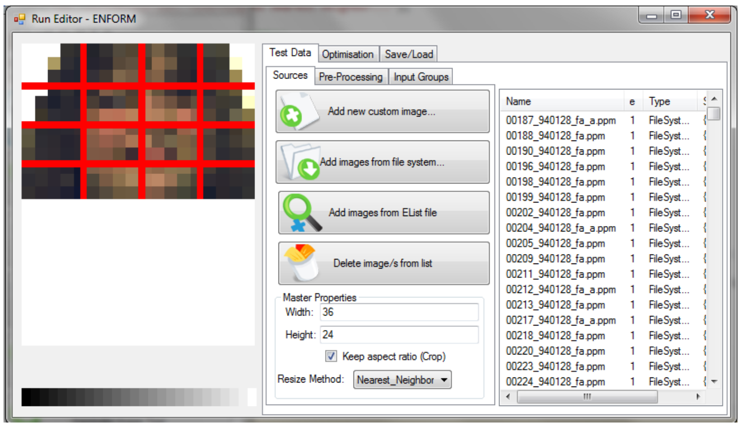
7 Conclusion

Back-propagation, for all its simplicity and antiquity still seems to be a very effective way to train neural networks with an unusually large number of weights. Particle Swarm Theory still needs improvements so that it is able to handle problems that are highly dimensional such as large neural networks. The next step would be to investigate ways to improve PSO and the work undertaken in the project would be a good benchmark for this type of reasearch.

8 Appendix

8.1 E.N.F.O.R.M.Research Tools

8.1.1 Run Editor



Master Properties
Width: 36
Height: 24
[✓] Keep aspect ratio (Crop)
Resize Method: Nearest_Neighbor

Name	e	Type	
00187_940128_fa_a.ppm	1	FileSyst...	{
00188_940128_fa.ppm	1	FileSyst...	{
00190_940128_fa.ppm	1	FileSyst...	{
00196_940128_fa.ppm	1	FileSyst...	{
00198_940128_fa.ppm	1	FileSyst...	{
00199_940128_fa.ppm	1	FileSyst...	{
00202_940128_fa.ppm	1	FileSyst...	{
00204_940128_fa_a.ppm	1	FileSyst...	{
00205_940128_fa.ppm	1	FileSyst...	{
00209_940128_fa.ppm	1	FileSyst...	{
00211_940128_fa.ppm	1	FileSyst...	{
00212_940128_fa_a.ppm	1	FileSyst...	{
00213_940128_fa.ppm	1	FileSyst...	{
00217_940128_fa_a.ppm	1	FileSyst...	{
00218_940128_fa.ppm	1	FileSyst...	{
00220_940128_fa.ppm	1	FileSyst...	{
00223_940128_fa.ppm	1	FileSyst...	{
00224_940128_fa.ppm	1	FileSyst...	{

Figure 11: The E.N.F.O.R.M. run editor.

The run editor is where each experiment is designed and the images processed in preparation for the optimiser. There are 2 main sections of the run editor: Test Data and Optimisation.

Sources The images are added either manually or from a set generated by the Face Explorer explained in 8.1.6 . The global size of the training images is selected here, along with the option to crop or stretch images of different aspect ratios.

Pre-processing Options can be selected here to apply processing to each image in the the training set before they are used. Stretch contrast and histogram correction are commonly used as they normalize the lighting slightly and bring out a better contrast which helps with smaller images.

Input Groups The way that the image is divided up into hidden nodes is defined here. A grid, which is type of input group used in this project, slices the image both horizontally and vertically and sends the input from each grid location to a hidden node. Horizontal and vertical input groups only cut the image up in one direction creating slices that are attached to hidden nodes.

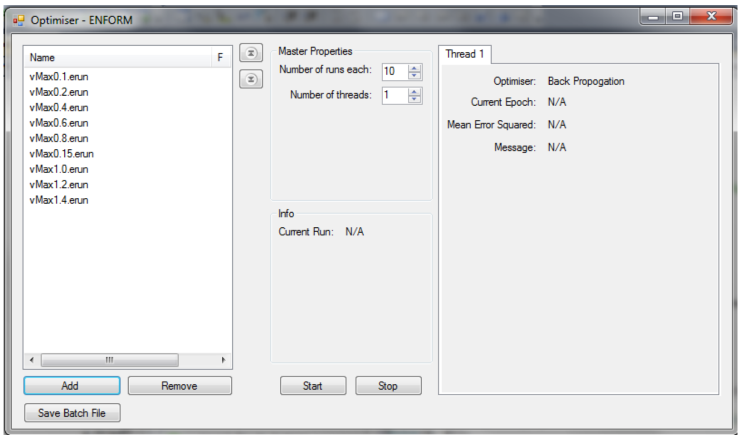
Global Parameters These parameters apply to both types of optimisation and simply impose limits to the execution of an experiment so that it doesn't run forever. The values chosen for these are further explained in 4.4. The buffer size is how many iterations are stored in memory before the experiment is paused and the data is flushed to disk.

Back-propagation These parameters control how back-propagation will function during a run. The learning rate parameters adjust the learning rate of the back-propagation algorithm and can be fixed or vary during the run according to a function of the number of performed iterations. The function of learning rate is further explained in 4.2.1. The jitter epoch is the number of iterations between applying jitter to the neural network and the noise limit is maximum amplitude of the noise applied to the neural network. Jitter is further explained in 4.2.1.

PSO These parameters control how PSO will function during a run. They map directly to the parameters mentioned in 4.2.2.

Save/Load This is where you save or load an experiment that has been designed. Upon saving the images are re sized, cropped, preprocessed and embedded in the experiment file so that it is portable.

8.1.2 Optimiser



Name	F
vMax0.1.erun	
vMax0.2.erun	
vMax0.4.erun	
vMax0.6.erun	
vMax0.8.erun	
vMax0.15.erun	
vMax1.0.erun	
vMax1.2.erun	
vMax1.4.erun	

Master Properties
Number of runs each: 10
Number of threads: 1

Info
Current Run: N/A

Optimiser: Back Propogation
Current Epoch: N/A
Mean Error Squared: N/A
Message: N/A

Figure 12: The E.N.F.O.R.M. optimiser.

This where the neural network is optimised and the results stored back in the experiment file.

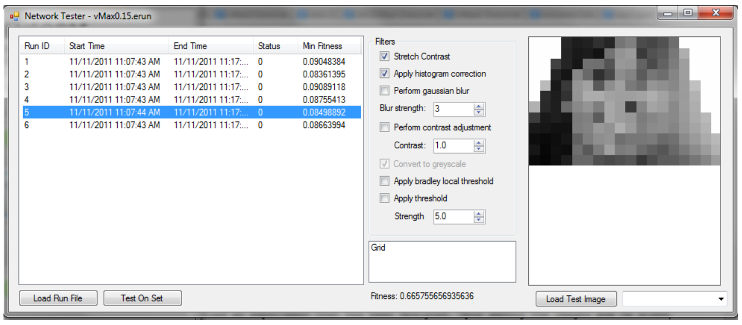
File list The experiment files are added here to be added to the queue and facilitate batch runs.

Master properties The number of runs each sets how many identical runs are added to the optimisation queue for each experiment. The number of threads sets how many simultaneous runs the optimiser can process. It is limited by the amount of available processors (or cores) available in the system as any more threads than this would result in no performance gain and create instability in the results due to unpredictable scheduling by the operating system. The number of threads is actually limited to 1 less than the available processors as 1 processor is reserved for co-ordinating the other threads and the UI thread.

Thread info These are mainly self-explanatory, each thread has it's own tab with info on the current run it is processing.

Save batch file The optimiser has the ability to save the file list and master properties to a batch file that can be imported by the command line version of the optimiser. This is used for running an optimiser on headless systems such as through ssh terminals.

8.1.3 Network Tester



Run ID	Start Time	End Time	Status	Min Fitness
1	11/11/2011 11:07:43 AM	11/11/2011 11:17:...	0	0.09048384
2	11/11/2011 11:07:43 AM	11/11/2011 11:17:...	0	0.08361395
3	11/11/2011 11:07:43 AM	11/11/2011 11:17:...	0	0.09089118
4	11/11/2011 11:07:43 AM	11/11/2011 11:17:...	0	0.08755413
5	11/11/2011 11:07:44 AM	11/11/2011 11:17:...	0	0.08498892
6	11/11/2011 11:07:43 AM	11/11/2011 11:17:...	0	0.08663994

Filters
- [✓] Stretch Contrast
- [✓] Apply histogram correction
- [ ] Perform gaussian blur
- Blur strength: 3
- [ ] Perform contrast adjustment
- Contrast: 1.0
- [✓] Convert to greyscale
- [ ] Apply bradley local threshold
- [ ] Apply threshold
- Strength 5.0

Grid

Fitness: 0.665755656935636

Figure 13: The E.N.F.O.R.M. network tester.

The network tester simple allows you to test the results of the optimisation runs on specific images or testing sets generated by the Face Explorer (see 8.1.6). This tool is mainly for quickly testing if a run was successful and does not generate the final results used in this report.

8.1.4 Results Analyser

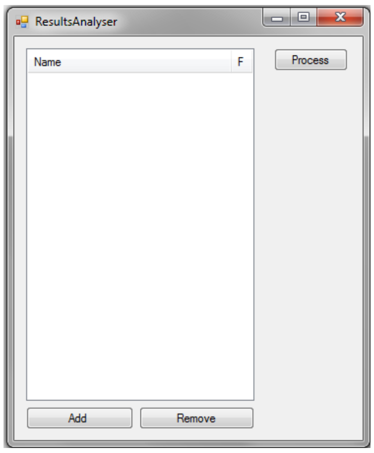
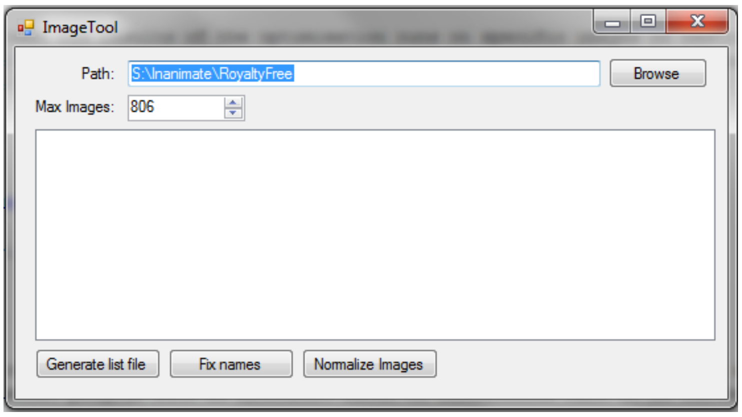


Figure 14: The E.N.F.O.R.M. results analyser.

The results analyser goes through every run in an experiment file and averages them, and dumps the result in a text file that can be opened in a spreadsheet program such as Microsoft Excel or LibreOffice Calc for further analysis.

8.1.5 Image Tool

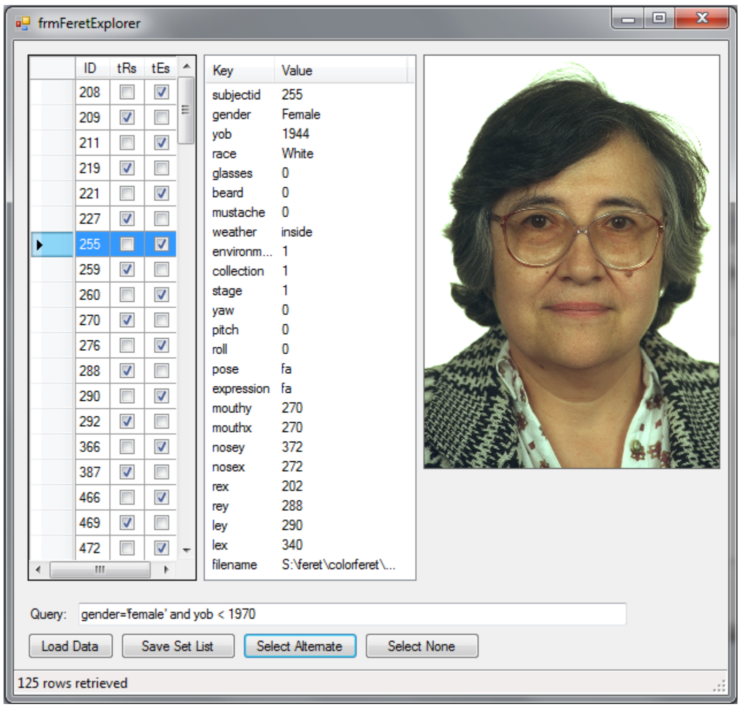


Path: S:\Inanimate\RoyaltyFree
Max Images: 806

Figure 15: The E.N.F.O.R.M. image tool.

This tool was used to generate training and testing sets from image categories, to check images and batch re size images. It also has other general purpose functions used in the project.

8.1.6 Face Explorer



ID	tRs	tEs
208	☐	✓
209	✓	☐
211	☐	✓
219	✓	☐
221	☐	✓
227	✓	☐
255	☐	✓
259	✓	☐
260	☐	✓
270	✓	☐
276	☐	✓
288	✓	☐
290	☐	✓
292	✓	☐
366	☐	✓
387	✓	☐
466	☐	✓
469	✓	☐
472	☐	✓

Key	Value
subjectid	255
gender	Female
yob	1944
race	White
glasses	0
beard	0
mustache	0
weather	inside
environm...	1
collection	1
stage	1
yaw	0
pitch	0
roll	0
pose	fa
expression	fa
mouthy	270
mouthx	270
nosey	372
nosex	272
rex	202
rey	288
ley	290
lex	340
filename	S:\feret\colorferet\...

Query: gender='female' and yob < 1970

125 rows retrieved

Figure 16: The E.N.F.O.R.M. face explorer.

Face explorer allows you to access the F.E.R.E.T. database and create training and testing sets.

The face browser on the left side allows you to select different subject IDs so you can view them and also any associated information. The left tick box designates that the image is part of the training set and the right tick box denotes the converse.
The query box is used to filter the subjects on their properties. The syntax is identical to an SQL where clause.
An example of such a query could be:

gender = 'female' and yob <1950

This would select all female subjects with a year of birth that is less than 1950. You can also use LIKE statements to wild card parameters.
For example:

gender like 'ma%'

This example would return all male subjects as the % symbol acts as a wild card which extends the 'ma' to male. As you type the face browser is updated dynamically.

The select alternate button selects every alternate check box so that half the faces are put into the training set and the other half into the testing set.

8.2 The Set of Non-Facial Images

32	Animals	26	Objects
40	Artwork	23	Outdoor Locations
35	Attractions	10	People
20	Birds	23	Plants and Nature
25	Buildings	7	Scenery
42	Cars	7	Sea Creatures
44	Cities	18	Settings
14	Education	10	Signs
12	Events	9	Space
9	Flowers	32	Sport and Recreation
35	Food and Drink	35	Structures
19	Fruit and Vegetables	9	Technology
32	Game Screen shots	65	Things that look like faces
46	Icons	14	Tools Machinery
10	Indoor Locations	11	Toys
14	Insects	37	Vehicles
21	Landscapes	12	Weather
8	Music		

8.3 Code

To impliment the velocity clamping there were some minor modifications done to the Standard PSO code.[2]

```
// For every dimension
for (d = 0; d < pb.SS.D; d++)
{
  //Acumulate average velocity of current run
  lAve += Math.Abs(PSOResult.SW.V[s].v[d]);
  //Count D*S
  vNum++;

  //The average velocity is calculated one iteration before
  if (iter > 1)
  {
    //Clamp velocity to average
    if (PSOResult.SW.V[s].v[d] > vMax)
    {
```

[2]The version of Standard PSO used can be found at https://github.com/firestrand/Standard-PSO

```
            PSOResult.SW.V[s].v[d] = vAve;
          }
          else if (PSOResult.SW.V[s].v[d] < -vMax)
          {
            PSOResult.SW.V[s].v[d] = -vAve;
          }
        }
        //Update position
        PSOResult.SW.X[s].x[d] = PSOResult.SW.X[s].x[d] + (PSOResult.
            SW.V[s].v[d]);
      }

      ...

      //After iteration complete calculate average
      vAve = lAve / (double) vNum;
```

8.4 List of figures and bibliography

List of Figures

References

[1] Scott E. Fahlman. An empirical study of learning speed in back-propagation networks. Technical report, 1988.

[2] Julius Gatune and Aryn Thomas. A look at facial recognition.

[3] M. Gori and A. Tesi. On the problem of local minima in backpropagation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14:76–86, 1992.

[4] D Nguyen and B Widrow. Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights. *International Joint Conference on Neural Networks*, 3:21–26, 1990.

[5] John Paul, T. Yusiong, and Prospero C. Naval. Training neural networks using multiobjective particle swarm optimization.

[6] P. Jonathon Phillips, Hyeonjoon Moon, Syed A. Rizvi, and Patrick J. Rauss. The feret evaluation methodology for face-recognition algorithms, 1999.

[7] Xiaorong Pu, Zhongjie Fang, and Yongguo Liu. Multilayer perceptron networks training using particle swarm optimization with minimum velocity constraints. In Derong Liu, Shumin Fei, Zeng-Guang Hou, Huaguang Zhang, and Changyin Sun, editors, *Advances in Neural Networks - ISNN 2007, 4th International Symposium on Neural Networks, ISNN 2007, Nanjing, China, June 3-7, 2007, Proceedings, Part III*, volume 4493 of *Lecture Notes in Computer Science*, pages 237–245. Springer, 2007.

[8] Henry A. Rowley, Student Member, Shumeet Baluja, and Takeo Kanade. Neural network-based face detection. *IEEE Transactions On Pattern Analysis and Machine intelligence*, 20:23–38, 1998.

[9] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2003.

[10] Ganesh K. Venayagamoorthy Salman Mohaghegi, Yamille del Valle and Ronald G. Harley. A comparison of pso and back-propagation for training rbf neural networks for identification of a power system with statcom. volume 5 of *Acoustics, Speech, and Signal Processing, 1991. ICASSP-91., 1991 International Conference on*, pages 3321–3324. Institute of Electrical and Electronics Engineers, 2005.

[11] Inc. The MathWorks. Neural network toolbox - limitations and cautions [of back propogation]. `http://www.kxcad.net/cae_MATLAB/toolbox/nnet/backp32a.html`.

[12] Xin Yao. Evolving artificial neural networks, 1999.

[13] Masood Zamani and Alireza Sadeghian. A variation of particle swarm optimization for training of artificial neural networks. *Computational Intelligence and Modern Heuristics*, 2010.

[14] Zongmei Zhang, Zhifang Sun, and Hiroki Tamura. Local linear wavelet neural network with weight perturbation technique for time series prediction. In *CSSE (4)'08*, pages 798–801, 2008.