



Norwegian University of
Science and Technology

Classifying Glyphs

Combining Evolution and Learning

Tiril Anette Langfeldt Rødland

Master of Science in Computer Science
Submission date: June 2011
Supervisor: Keith Downing, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Problem Description

The goal of this dissertation is to use artificial neural networks to classify the writing system of previously unseen glyphs, e.g. Runes and Hieroglyphs. The artificial neural networks shall be trained using both evolution and back-propagation learning. Combinations of these methods will be investigated, and compared to the pure methods.

Assignment given: 17. January 2011

Supervisor: Keith Linn Downing

Abstract

This dissertation investigates the classification capabilities of artificial neural networks (ANNs). The goal is to generalize over the features of a writing system, and thus classify the writing system of a previously unseen glyph. The complexity of the problem necessitates a large network, which hampers the tuning of the weights.

ANNs were created using three different hybrids of back-propagation (BP) learning and evolution, and a pure BP algorithm for comparison. The purpose was to find the method best suited for this kind of generalization and classification networks.

The results suggest that ANNs are able to generalize enough to solve the classification task, but it is depending on the weight tuning algorithm. A pure BP algorithm is preferable to any of the hybrid algorithms, due to the size of the ANN. This algorithm had both the best classification results and the fastest runtime, in addition to the least complex implementation.

Preface

This master dissertation is written at the Department of Computer and Information Science, NTNU, during the spring semester of 2011. It is based on the prestudy project (Rødland, 2010), written during the autumn semester of 2010.

The original aim for the project was to investigate how well an artificial neural network (ANN) is able to generalize. ANNs would be used to classify the writing system of an unseen glyph, by generalizing over the shapes and features common for the writing systems. During the research on the prestudy project, I tested ANNs created using learning (a back-propagation (BP) algorithm), and using evolution (a genetic algorithm (GA)). It turned out that only the ANN created by BP was able to solve the classification problem. Inspired by this, the next step was to investigate whether hybrids of BP learning and evolution are better suited than the pure algorithms.

I wish to thank my supervisor Professor Keith L. Downing at the Department of Computer and Information Science, Norwegian University of Science and Technology, for his invaluable assistance.

Special thanks are given to John A. Bullinaria, for his immense help in debugging, and for sharing of his infinite wisdom regarding neural networks.

Gratitude is given to Arne Dag Fidjestøl and Aslak Raanes in the System Administration Group, Department of Computer and Information Science, for giving me access to hardware on which I could run the evolution.

I am also very grateful to Cyrus Harmon, Edi Weitz, David Lichteblau, and all the other Common Lisp programmers out there, whose libraries saved me from months of extra work.

Trondheim, May 2011

Tiril Anette Langfeldt Rødland

Contents

Abstract	ii
Contents	vii
List of Figures	x
List of Tables	xi
List of Algorithms	xiii
Abbreviations and Acronyms	xv
Chapter 1 : Introduction	1
1.1 Background and Motivation	1
1.1.1 Writing Systems	1
1.1.2 Glyph Classification	2
1.2 Problem Definition	3
1.2.1 Research Questions	4
1.3 Outline	4
Chapter 2 : Background	5
2.1 Writing Systems	5
2.2 Training Artificial Neural Networks	8
2.2.1 Genetic Algorithm	8
2.2.2 Back-Propagation	9
2.2.3 Weight Evolution	12
2.2.4 Back-Propagation/Genetic Algorithm	17
2.2.5 Genetic Algorithm/Back-Propagation	18
2.2.6 Other Hybrids	18
2.2.7 Tricks and Details to Improve Algorithms	23
Chapter 3 : Methodology	25
3.1 Glyphs and Writing Systems	25

3.1.1	Runic	27
3.1.2	Ugaritic	27
3.1.3	Hebrew	28
3.1.4	Arabic	28
3.1.5	Canadian Aboriginal Syllabics	29
3.1.6	Hiragana	29
3.1.7	Thai	30
3.1.8	Devanagari	30
3.1.9	Han	30
3.1.10	Egyptian Hieroglyphs	32
3.2	Artificial Neural Network Design	33
3.2.1	Nodes	33
3.2.2	Architecture	34
3.3	Algorithms	36
3.3.1	Genetic Algorithm	37
3.3.2	Back-Propagation	39
3.3.3	Weight Evolution	42
3.3.4	Back-Propagation/Genetic Algorithm	44
3.3.5	Genetic Algorithm/Back-Propagation	45
Chapter 4 : Results and Discussion		47
4.1	Algorithm Comparison	47
4.1.1	Training	47
4.1.2	Testing	50
4.2	Writing System Comparison	51
4.2.1	Comparing Classification Results with Glyph Appearance	52
4.3	Potential Bias Problems	53
4.3.1	Test with 25 Runs	53
4.3.2	Test with 3 Runs	55
4.3.3	Conclusion	56
Chapter 5 : Conclusion		57
5.1	Concluding the Research Questions	57
5.1.1	RQ1: Generalization	57
5.1.2	RQ2: Method	58
5.2	Future Work	60
5.2.1	Improving the Generalizing ANN	60
5.2.2	Improving the Glyph Classification	60
References		63
Software Library		69
Appendix A : Unicode Charts		A-1

Appendix B: Result Graphs	B-1
B.1 How to Read the Graphs	B-1
B.1.1 Training Graphs	B-1
B.1.2 Testing Graphs	B-2
B.2 Main Results	B-2
B.2.1 Results Sorted by Algorithm	B-2
B.2.2 Results Sorted by Phase	B-7
B.3 Different Number of Runs	B-9
B.3.1 Maximum 25 Runs	B-9
B.3.2 Maximum 3 Runs	B-13
B.4 Genetic Algorithm	B-15

List of Figures

2.1	Back-propagation ANN	10
2.2	GA wavelet-BP	21
3.1	Pixel comparison using pixel value	27
3.2	Han evolution	31
3.3	ANN: From Unicode to output values	35
3.4	GA flowchart	37
3.5	BP flowchart	40
3.6	WE flowchart	42
3.7	BP/GA flowchart	44
3.8	GA/BP flowchart	46
4.1	Training phases with all available data	48
4.2	Testing phases with all available data, in bar view	49
4.3	Testing phases with all available data, in matrix view	50
4.4	Testing phases with maximum 25 runs, in bar view	54
4.5	Testing phases with maximum 3 runs, in bar view	55
A.1	Runic	A-2
A.2	Ugaritic	A-3
A.3	Hebrew	A-4
A.4	Arabic	A-5
A.5	Canadian Aboriginal Syllabics	A-6
A.6	Hiragana	A-7
A.7	Thai	A-8
A.8	Devanagari	A-9
A.9	Han	A-10
A.10	Egyptian Hieroglyphs	A-11
B.1	BP results	B-3
B.2	WE results	B-4
B.3	BP/GA results	B-5

B.4	GA/BP results	B-6
B.5	Training phases with all available data	B-7
B.6	Testing phases with all available data, in bar view	B-8
B.7	Testing phases with all available data, in matrix view	B-9
B.8	Training phases with maximum 25 runs	B-10
B.9	Testing phases with maximum 25 runs, in bar view	B-11
B.10	Testing phases with maximum 25 runs, in matrix view	B-12
B.11	Training phases with maximum 3 runs	B-13
B.12	Testing phases with maximum 3 runs, in bar view	B-14
B.13	Testing phases with maximum 3 runs, in matrix view	B-15
B.14	Genetic algorithm	B-16

List of Tables

3.1	Writing systems	26
3.2	Parameters used by the algorithms.	36
3.3	The gene	38
4.1	Ratio of correct classification	51
B.1	GA parameters	B-16

List of Algorithms

2.1	Batch back-propagation	24
3.1	GA algorithm	37
3.2	BP algorithm	40
3.3	WE algorithm	43
3.4	BP/GA algorithm	45
3.5	GA/BP algorithm	45

Abbreviations and Acronyms

ANN artificial neural network

ATHENA the analysis tool for heritable and environmental network associations

BP back-propagation

BP/GA back-propagation/genetic algorithm

CG conjugate gradients

DE differential evolution

EC evolutionary computation

EEC evolving efficient connections

EM expectation-maximization

GA genetic algorithm

GA/BP genetic algorithm/back-propagation

GAA genetic annealing algorithm

IPA International Phonetic Alphabet

IR infrared

LH1-DECG Lamarckian hybrid of DE and CG: approach 1

LH2-DECG Lamarckian hybrid of DE and CG: approach 2

NEEC neural network enzyme classification

PIM plastic injection molding

SA simulated annealing

SAR synthetic aperture radar

SSE sum squared error $E = \frac{1}{2} |T - Y|^2$

TSS total sum squared error

WE weight evolution

XOR exclusive or

CHAPTER 1

Introduction

This chapter will hopefully describe the motivation behind the dissertation topic, as well as the ultimate goal and research questions. In the end of the chapter, the outline for the dissertation is presented.

1.1 Background and Motivation

This dissertation discusses glyph classification, i.e. classification of the writing system of a previously unseen glyph. A glyph is an image used to visually represent a sound in a language (FOLDOC, 1998a), while a writing system can be seen as the set of glyphs used to represent the writing of one or more human languages (FOLDOC, 1998b).

1.1.1 Writing Systems

33 000 years have passed since the first *Homo sapiens* discovered that they could draw recognizable pictures on the wall of the Chauvet cave in France (Dehaene, 2009). Since then, writing has evolved from pictures of animals and nonfigurative shapes, to easy-to-write shapes that represent ideas or sounds. The original pictures have evolved to different shapes in different cultures, but one can still see the original meaning in some of the new characters, e.g. *A* and α , which are simplifications of an ox (*aleph* in Greek). The horns and head are visible in both glyphs; however, *A* is upside down, and α is rotated right.

Similarly, 牛 is the Chinese character for *ox*, or *cow*. So even though both *A* and 牛 have developed from the image of an ox, the resulting glyphs are very different. 牛 has more in common with 本 than *A*, even though 本 means *book*, which is not at all related to bulls. However, some writing systems that are closely related

have similar glyphs. For example, *A* can be found in the European alphabets of Latin, Cyrillic, and Greek; as a capital *a*, *a* and α , respectively. This is the case for writing systems all over the world. Some are far apart and look nothing alike, while others are more closely related, with similar glyphs. (Ager, 2011)

Nevertheless, there are numerous visual features that all writing systems have in common, e.g. contrasted contours and an average of three strokes per character. They also have a reduced set of shapes that frequently occur in the glyphs; these shapes characterize the writing system. Some have a set mainly consisting of straight lines (Runic), while some have mostly circles (Shan). Several shapes are common in multiple writing systems, e.g. lines and curves, but the frequency and composition of the shapes tend to differ between the writing systems. (Dehaene, 2009; Ager, 2011)

Humans are usually able to separate between different writing systems, and recognize the writing system of an unseen glyph if one already is familiar with other glyphs from said writing system. This trait is probably due to the neural network in our brain, which generalizes over the set of seen glyphs, and extracts the common shapes and features (Dehaene, 2009). As a natural neural network is able to do this, an artificial neural network (ANN) ought to be able to as well.

1.1.2 Glyph Classification

ANNs are able to recognize glyphs. Character recognition is a much researched area, and an important part of general pattern recognition (Bishop, 1996). Character recognition is the task of recognizing a character the system already has been in contact with; in worst case the character has additional noise (Bishop, 1996). The noise recognition capabilities are used to e.g. read handwritten characters or characters written with different fonts.

One could solve the glyph classification problem by running all glyphs through the network, and have the network recognize all glyphs and remember their writing system. However, this solution is not that exciting, precisely because ANNs *are good at* character recognition (Bishop, 1996). It would also be a suboptimal solution; not because it is a difficult task, but because the network would have to train on huge amounts of glyphs. It should train on every glyph of every writing system, as it would fail completely each time an unseen glyph appeared.

To both avoid the large training sets and find a challenging task, the ANNs must learn to generalize. That is, they must learn to look at the writing system as a whole, and generalize over the common shapes and features. It is thus important to completely separate between the training set and the test set, such that no glyph in the test set can have been seen before, and thus recognized.

It is this generalization that makes glyph classification more difficult than normal character recognition, as different versions of the same character are more similar than different characters from the same writing system. As can be seen in Appendix A, the glyphs within a writing system can look quite different. Even though the shapes and features are common, they must look sufficiently different from each other for it to be possible to differentiate between them while reading. Consequently, the characters within a writing system have their black and white

pixels in different patterns, and trigger thus different neurons. Variations of the same character will have most of their pixels in the same place, and triggering mostly the same neurons.

1.2 Problem Definition

The main goal of this dissertation is not to find a solution that can classify the writing system of a glyph. As the writing system tend to be known by context, this is not a very useful task. The main goal is to investigate *how well ANNs are able to generalize*. The focus will thus not be on creating a system as good as possible, but to make the generalization as good as possible, even if that results in a suboptimal solution.

To investigate the generalization capabilities of ANNs, one should create different ANNs using different methods, as one method might solve the problem better than another. The main focus is on learning and evolution, as those are common methods for training ANNs (Floreato and Mattiussi, 2008; De Jong, 2006).

The prestudy for this dissertation (Rødland, 2010) compared the use of pure learning and pure evolution on this generalization task. It was concluded that a genetic algorithm (GA) in itself is unable to solve the glyph classification problem, as it was too memory consuming and time demanding to achieve acceptable results. The back-propagation (BP) algorithm, on the other hand, solved the glyph classification problem satisfactorily.

This dissertation is based on the hypothesis that evolution might work better if it is combined with another method. Hopefully, this would help solve the time and memory problems. To investigate this hypothesis, three hybrid algorithms combining evolution and BP learning are compared. They are also compared to the pure BP algorithm, to investigate whether BP with evolution is preferable to pure BP.

GA is not in this comparison due to the unsatisfactory results achieved in Rødland (2010); however, the results from Rødland (2010) are available in Section B.4. The hybrids can thus not be directly compared to the pure GA. This is not seen as a problem; if their results are as bad as those of GA, they are — for all intents and purposes — incapable of solving the glyph classification problem. As there is at least one algorithm that does solve the problem (BP), the internal rank order of those that do not solve it, is not that important.

The three hybrid algorithms are weight evolution (WE), back-propagation/genetic algorithm (BP/GA), and genetic algorithm/back-propagation (GA/BP). WE is a batch back-propagation algorithm that has one generation of evolution within every iteration of learning. This is to escape from local minima and improve low performance nodes. The other two use both online BP and a GA. BP/GA uses BP to reduce the search space, and finds the final solution using a GA. GA/BP does the opposite; it reduces the search space using a GA, and then it applies the BP.

1.2.1 Research Questions

As the main focus of this dissertation is the generalization capabilities of ANNs, this will constitute the first and most important research question, RQ1. Whether generalization is possible or not, some algorithms are probably better suited than others. The secondary research question is thus which method is the best suited for this problem, be it learning, evolution, or a hybrid of the two.

There are thus two main research questions that this dissertation will try to answer:

RQ1 Are ANNs able to generalize over glyphs of many different writing systems, and correctly classify the writing system of an unseen glyphs?

RQ2 Which training method would best solve RQ1; learning, evolution, or a hybrid of learning and evolution?

1.3 Outline

This dissertation is structured as follows.

Chapter 1 introduces the problem and the research goals.

Chapter 2 gives an introduction to writing systems in general. It also presents previous research on algorithms for training ANNs, focusing on hybrid algorithms.

Chapter 3 introduces the chosen writing systems. In addition, the design of the ANN is described, as well as the algorithms used in the system.

Chapter 4 presents the results, along with a discussion of both the algorithms and the writing systems. It also discusses some potential bias problems.

Chapter 5 concludes the dissertation, along with a discussion of future work.

Appendix A includes the Unicode charts from the writing systems used. This is to get a feeling of how the glyphs look, to easier compare the glyphs within the writing system, as well as compare different writing systems. This will hopefully help the reader see why some writing systems are easier to classify than others.

Appendix B includes all the result graphs, sorted by both algorithm and phase. In addition, the results from the GA from Rødland (2010) are included, for comparison purposes.

CHAPTER 2

Background

This chapter consists of two parts; Section 2.1 describes the different type of writing systems, and Section 2.2 presents different hybrid algorithms for training of artificial neural networks (ANNs). Focus is on the algorithms this dissertation is based on, but also other hybrids are discussed.

2.1 Writing Systems

There are different kinds of writing systems, depending on the type of sound each glyph represents. Below is the classification used by The Unicode Consortium (2011, Ch. 6).

Alphabet Alphabets are writing systems that consist of letters, which can be either a consonant or a vowel. Both have the same status as letters, giving a homogeneous collection of glyphs.

Alphabets are quite common in the Western world. The most well-known is probably the Latin alphabet, but also Greek, Cyrillic, Coptic, and Runic are alphabets. The name comes from the first two letters of the Greek alphabet; *alpha* (α) and *beta* (β).

There is often varying correspondence between letters and sounds in alphabets. The Latin alphabet is a good example, as it is used to write a huge variety of languages. Some of these languages, e.g. Italian and Finnish, have a high correlation between letters and sound, meaning that the pronunciation of any unknown word is clear based on the spelling of the word, and vice versa. Other languages, e.g. English, have evolved and incorporated words from many other languages. This results in a spelling and pronunciation that is highly complex and difficult to

determine. However, there are also phonetic alphabets, with International Phonetic Alphabet (IPA) as the best known. These alphabets are designed to have a one-to-one correspondence between sound and letter.

Abjad In abjads, all the glyphs are consonants. The vowels are either completely missing from writing, or only indicated by specific marks on the consonants. For example, Arabic and Hebrew are seldom written with vowels. When they are written, they are indicated by points, or harakat. These are diacritic dots and other marks, placed either above or below the consonants.

Semitic languages are often abjads, as these languages have a word structure that is best suited for consonantal writing. These include Arabic, Hebrew, and Syriac. All abjads — with the exception of Ugaritic — are written from right to left (Ager, 2011).

The name *abjad* is from the first four letters of Arabic: *alef, beh, jeem, dal*.

Syllabary The glyphs in a syllabary consist of minimum one consonant and one vowel. Syllabaries are named as such because each glyph is called a *syllabary* instead of a *letter*.

The Japanese kana systems — Hiragana and Katakana — are both syllabaries. The glyphs there describe syllables like e.g. *ka* (か), *ki* (き), *ku* (く), *ke* (け), *ko* (こ), *ra* (ら), and *ri* (り). Here, each glyph is unique in shape, such that か and き look nothing alike, even though they both start with the sound *k*. This feature can be found in other syllabaries as well, e.g. Cherokee and Yi.

Other syllabaries, like the Canadian Aboriginal Syllabics, do have similar *ka* and *ki*. There, the shape of the glyph decides the consonant, while the rotation decides the vowel.

Abugida Abugidas mix the features of alphabets and syllabaries. That is, each consonant has its own inherent vowel, usually an *a*. If another vowel follows a consonant, the inherent vowel is overridden. For example, an *i* that follows a *ka*, results in *ki*, not *kai*. Some vowel glyphs (matras) are subordinate to the consonant letters — overriding their inherent vowel — while other vowels can be used as independent vowels, e.g. at the beginning of words. Abugidas typically have a special glyph, the halant, that removes the inherent vowel. A halant following a consonant will thus give a bare consonant sound.

The Devanagari script — used to write e.g. Hindi and Sanskrit — is an abugida. So is Thai, which also can be used to write Sanskrit. The Ethiopic script is also an abugida, and it is the first four letters of this script that have given abugidas their name: *alf, bet, gaml, and dant*. The vowels (*ä, u, i, a*) are in the traditional order in the Ethiopic script charts.

Interestingly, the Ethiopic script is not a true abugida. It is classified as such because the basic character for each series of consonants has an inherent vowel. However, other features — like matras and halant — are missing. The Ethiopic script is derived from early Semitic scripts, and was originally an abjad. However,

today it is treated mainly as a syllabary, due to its traditional presentation and encoding.

Logosyllabary Logosyllabaries do not represent only the sound of a language, but the meaning behind the words.

The Han script — which is used to write Chinese — is a logosyllabary. These glyphs are borrowed by other East Asian languages, e.g. Japanese and Korean. As a consequence, it is possible to understand written Chinese when you know Japanese, even though the spoken languages are nothing alike. A Han glyph is called *hànzì* in Chinese, *kanji* in Japanese, and *hanja* in Korean. In all three languages it is written as 漢字.

Japanese is special, in that it uses four different writing systems in the same composite system (Banno et al., 1999). That is, kanji (logosyllabary) for things and meanings, Hiragana (syllabary) for e.g. grammatical postfixes and particles, Katakana (syllabary) for e.g. foreign words, and Latin (alphabet) for rōmaji. Rōmaji is a transliteration of the sound, for use when the other systems will not be understood, e.g. when working with computers and foreigners.

Because of this separation, the kanji is used only for meaning, not sound. Each kanji can have many different sounds connected to it. As a result, the same sequence of kanji can be translated to different strings of Hiragana, depending on the context. 日曜日, meaning *Sunday*, is pronounced *nichiyoubi*, にちようび. Note that the first and the second 日 have different sounds. This is because they have different meanings; the first 日 means *Sun*, while the second means *day*. (曜 means *weekday*.) The basic meaning is the same, however, as the day can be seen as a function of the Sun. Similarly, 月 means both *Moon* and *month*. When it comes to logosyllabaries, context is everything. A Hiragana ㇿ, on the other hand, will always be a *te*.

Egyptian Hieroglyphs can double as both kanji and kana. While the majority of the Hieroglyphs are logograms, there is also a subset of these that behaves like an alphabet. Other glyphs represent sequences of consonants. There are also some glyphs that function as a determinative, i.e. an unspoken character that is placed at the end. Their purpose is to give the reader some context, a general idea of the meaning. In other words, Hieroglyphs are complicated. For simplicity they are classified as a logosyllabary by The Unicode Consortium (2011), while there actually are many differences. This is an overall problem with writing systems, as they were not designed to fit a specific type.

One can take the glyph \square (*pr*) as an example (Collier and Manley, 2003). This looks like the floor plan of a simple one-room house, and is thus the symbol used for *house*. \square means an actual house, as \uparrow is a sign used to indicate that this indeed is a logogram. When used in $\square \Delta$ (*pr*), on the other hand, \square has nothing to do with houses. Similarly, \ominus (*r*) does no longer mean *mouth*. \square is now used for the sound *pr*, which means *to go out, leave*. The \ominus is there to clarify the reading of \square , not to add an extra *r*. The walking legs at the end, Δ , are used as a determinative, to show the basic idea of *movement*.

2.2 Training Artificial Neural Networks

The performance of an ANN depends on the weight values. These can be manually configured in very small networks. However, this task is too complex for larger networks, necessitating a training algorithm.

This section describes algorithms used for ANN training. These algorithms include evolution, learning, and many evolution-learning hybrids. The first five algorithms are used in the glyph classification problem, while other hybrid algorithms are introduced in Section 2.2.6.

2.2.1 Genetic Algorithm

Evolutionary computation (EC) is a search method deeply inspired by the concept of evolution (Darwin, 1859). A genetic algorithm (GA) is an evolutionary method, designed for bit vector individuals (Downing, 2009). A bit vector genotype gives the most freedom when translating the genotype to a phenotype; a bit vector can represent almost anything, and is thus a good choice when evolving an ANN (Whitley et al., 1990). That points to the use of a GA, and thus also the use of both mutation and crossover, full generational replacement as adult selection, a uniform parent selection, and a fitness proportionate mate selection (Downing, 2009).

There are different approaches to evolving ANNs; one can e.g. use a predefined topology and only evolve the weights, or one can evolve the topology as well (Jones, 2009; Dewri, 2003).

Since this ANN shall be used by multiple algorithms, the topology ought to be predefined. That would make the implementation less complicated and more modular. However, it is still possible to slightly adapt the network. Shi and Wu (2008) designed evolving efficient connections (EEC), a method to evolve both connection weights and a switch to turn the weight on or off. Otherwise, one must wait for all the bits to become 0 for the connection to be removed; with EEC just one bit needs to be turned off.

A GA has already been used on the glyph classification problem (Rødland, 2010). The results (Section B.4) suggested that the ANN necessary to solve the problem was too large for evolution to handle; the runtime was extensive and so was the required memory. Only four runs were completed, and the results achieved were not impressive. As can be seen in the test graph (Figure B.14b on page B-17), there seemed to be no detectable difference between the writing systems. It was concluded that evolution was too demanding with respect to both time and resources to be an acceptable solution for such a large ANN.

2.2.1.1 Algorithm

According to De Jong (2006), a GA is as follows:

Step 0: Initialization Randomly generate a population of m parents.

Step 1: Fitness Compute and save the fitness $u(i)$ for each individual i in the current parent population.

Step 2: Selection Define selection probabilities $p(i)$ for each parent i so that $p(i)$ is proportional to $u(i)$.

Step 3: Reproduction Generate m offspring by probabilistically selecting parents to produce offspring.

Step 4: Surviving Let only the offspring survive to the next generation.

Step 5: Continue Go to Step 1, until the fitness of the best individual is high enough.

2.2.2 Back-Propagation

Using supervised learning, the weights of the ANN are learned by exploiting the difference between the output of the network and the known solution.

For each pattern μ of the M training patterns, there is a pair of input vectors \vec{x}^μ and desired output vectors \vec{t}^μ . The goal is to find the set of synaptic weights such that the actual output \vec{y}^μ of a two-layered network is as close as possible to the desired output \vec{t}^μ , for all of the M patterns.

The weights of the network should change gradually, until it is performing acceptably. The performance can be described using an error function, e.g. the mean quadratic error between the desired and the actual output (Equation (2.1)). Just as the output \vec{y}^μ , the error is depending on the synaptic weights. The error can thus be reduced by changing the weights accordingly.

$$E_W = \frac{1}{2} \sum_{\mu} \sum_i (t_i^\mu - y_i^\mu)^2 = \frac{1}{2} \sum_{\mu} \sum_i \left(t_i^\mu - \sum_j w_{ij} x_j^\mu \right)^2 \quad (2.1)$$

The weight change is given by a learning rule. Differentiating Equation (2.1) with respect to the weights gives the learning rule in Equation (2.2), where η is the learning rate (Wagstaff, 2008; Floreano and Mattiussi, 2008). This equation is known both as the *Widrow-Hoff rule* — after its authors Widrow and Hoff (1960) — and as the *delta rule*, because of the difference $\delta = t - y$. It is often written as simply $\Delta w_{ij} = \eta \delta_i x_j$.

$$\Delta w_{ij} = \eta (t_i - y_i) x_j \quad (2.2)$$

This Δw_{ij} is then added to the weight w_{ij} , either *online*, i.e. after each and every pattern, or in *batch mode*, i.e. adding the accumulated weight change after all the M training patterns have been run through the net.

While the delta rule works fine for training patterns that are linearly separable, it does not work for more complex patterns, e.g. exclusive or (XOR). The XOR

function needs hidden layers, necessitating a more complex version of the delta rule. The solution — the *generalized delta rule* — was presented by Rumelhart et al. (1986). This algorithm, best known as *back-propagation of error*, propagates the error back through the network, from the outer layer back through all the hidden layers, and to the input layer. It can thus handle an ANN with an arbitrary number of nodes and layers.

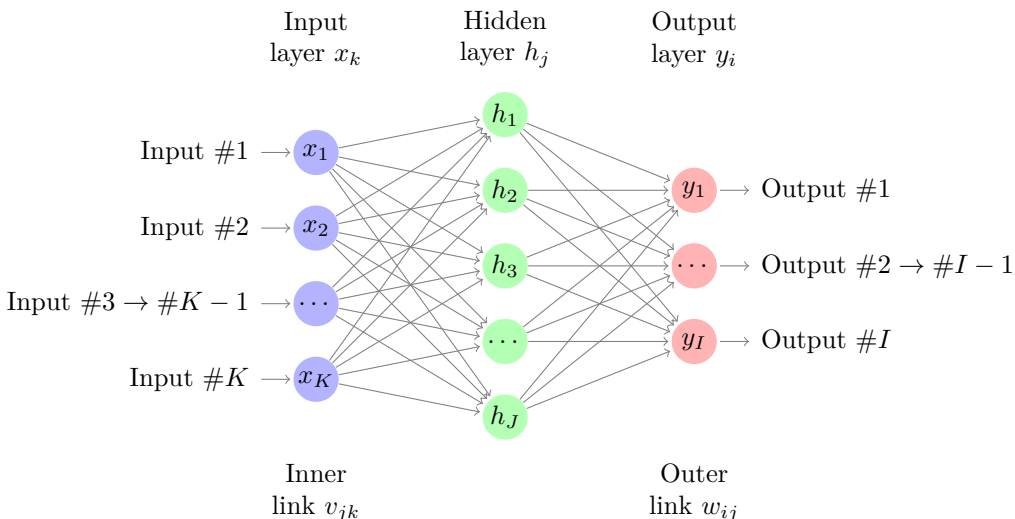


Figure 2.1: An ANN with the back-propagation syntax. x_k is the input layer, with K nodes. Similarly, h_j and y_i are the hidden layer with J nodes and the output layer with I nodes, respectively. v_{jk} is the link between the input layer and the hidden layer. Each node in the input layer is connected to every node in the hidden layer, so the link consists of $k \times j$ connections. Similarly, w_{ij} is the link between the hidden layer and the output layer.

A back-propagation (BP) algorithm has also been tested on the glyph classification problem before (Rødland, 2010). Good results were achieved within reasonable time, suggesting that BP — unlike the GA — is capable of solving the glyph classification problem.

2.2.2.1 Algorithm

The BP algorithm given by Floreano and Mattiussi (2008) for online weight updating — using the syntax from Figure 2.1 — is as follows:

Step 0: Initialization Initialize all weights, v_{jk} and w_{ij} , to random values close to 0. Set the values of the input nodes to the current training pattern s :

$$x_k^\mu = s_k^\mu \quad (2.3)$$

Step 1: Feed-forward Compute the values of the hidden nodes, based on the input nodes and the link from the input to the hidden layer:

$$h_j^\mu = \Phi \left(\sum_k v_{jk} x_k^\mu \right) \quad (2.4)$$

Compute the values of the output nodes, based on the hidden nodes and the link from the hidden to the output layer:

$$y_i^\mu = \Phi \left(\sum_j w_{ij} h_j^\mu \right) \quad (2.5)$$

The function $\Phi(\cdot)$ is the sigmoid function:

$$\Phi(a_i) = \frac{1}{1 + e^{-ca_i}} \quad (2.6)$$

Step 2a: Compute the error in the outer layer Compute the delta error for each output node, based on the hidden nodes and the link from the hidden to the output layer, as well as the difference between the expected and the actual output:

$$\delta_i^\mu = \Phi' \left(\sum_j w_{ij} h_j^\mu \right) (t_i^\mu - y_i^\mu) \quad (2.7)$$

It can be noted that the derivative of the sigmoid function can be expressed in terms of the output of the sigmoid function (Wagstaff, 2008):

$$\Phi' \left(\sum_j w_{ij} h_j^\mu \right) = y_i^\mu (1 - y_i^\mu) \quad (2.8)$$

As a result, the delta error can be written solely depending on the actual and the expected output:

$$\delta_i^\mu = y_i^\mu (1 - y_i^\mu) (t_i^\mu - y_i^\mu) \quad (2.9)$$

Step 2b: Compute the error in the inner layer Compute the delta error for each hidden node, based on the input nodes, both links, and the delta error for the output nodes:

$$\delta_j^\mu = \Phi' \left(\sum_k v_{jk} x_k^\mu \right) \sum_i w_{ij} \delta_i^\mu \quad (2.10)$$

Thanks to the derivative of the sigmoid function, the error can be written independently of the input:

$$\delta_j^\mu = h_j^\mu (1 - h_j^\mu) \sum_i w_{ij} \delta_i^\mu \quad (2.11)$$

Step 3: Compute the weight changes Compute the changes in synaptic weights, based on the delta errors and the output values of the first two layers:

$$\Delta w_{ij}^\mu = \delta_i^\mu h_j^\mu \quad (2.12a)$$

$$\Delta v_{jk}^\mu = \delta_j^\mu x_k^\mu \quad (2.12b)$$

Step 4: Update the weights Update the weights by adding the changes to each weight, multiplied with the learning rate η :

$$w_{ij}^t = w_{ij}^{t-1} + \eta \Delta w_{ij}^\mu \quad (2.13a)$$

$$v_{jk}^t = v_{jk}^{t-1} + \eta \Delta v_{jk}^\mu \quad (2.13b)$$

Step 5: Continue Go to Step 1, until the total sum squared error (TSS) is sufficiently small. This error (Equation (2.14)) is computed over all the output nodes i , for all the training patterns μ .

$$TSS = \frac{1}{M} \sum_{\mu} \left(\frac{1}{N} \sum_i (t_i^\mu - y_i^\mu)^2 \right) \quad (2.14)$$

2.2.3 Weight Evolution

Ng and Leung (2000) proposed a weight evolution (WE) algorithm that evolves weights within the BP algorithm. There are two reasons for doing this; (1) to achieve faster convergence, and (2) to escape from local minima.

2.2.3.1 Overview

The algorithm is based on BP. Each epoch of BP will probably reveal some output nodes with an error significantly above average. The WE algorithm will select and change certain weights connected to these nodes. For each of the nodes, the incoming weights will be duplicated and mutated, producing several offspring. These weights will be updated, and the error recalculated. The original set of weights will then be replaced by the fittest weight set available; either one of the offspring, or the original. A new epoch of BP will start, with the new weights.

In case a local minimum is reached, the weight evolution between the input and the hidden layer is initiated. When caught in a local minimum, some input patterns will perform significantly worse than others. For these problematic input pattern, the weights between a particular hidden node and its input nodes will be evolved. This will continue for as long as the network is caught in local minima.

2.2.3.2 Details

According to the syntax of Ng and Leung (2000), x is the input pattern, h and y are the output of the hidden layer and the output layer respectively, and t is the target output. A three-layered network has K input nodes, J hidden nodes, and I

output nodes (Figure 2.1 on page 10). The weights between the input and hidden layer — the inner link — is represented using v , while the outer link — between the hidden and output layer — is written as w .

Using this syntax, the output of the j th node in the hidden layer — assuming input pattern p — is given by Equation (2.15), and the i th node in the output layer is given by Equation (2.16). In both of these, $\Phi(\cdot)$ is the sigmoid function (Equation (2.17)).

$$h_{pj} = \Phi \left(\sum_{k=1}^K v_{jk} x_{pk} \right) \quad (2.15)$$

$$y_{pi} = \Phi \left(\sum_{j=1}^J w_{ij} h_{pj} \right) \quad (2.16)$$

$$\Phi(n) = \frac{1}{1 + e^{-n}} \quad (2.17)$$

The sum of squared error for the system is defined as Equation (2.18).

$$E = \frac{1}{2} \sum_{p=1}^P \sum_{i=1}^I (t_{pi} - y_{pi})^2 \quad (2.18)$$

Weight evolution in the outer link The weight evolution in the outer link is used for output neurons with especially high error. For example, i^* ($1 \leq i^* \leq I$) is an output neuron for which the output error is higher than average squared error (Equation (2.19)). As the current weights are suboptimal, the weights w_{i^*j} connected to this node ought to be evolved. By changing these weights and keeping the others, the error of i^* will be reduced while the error of the other output nodes will be preserved. The total error will thus be reduced, speeding up the convergence.

$$\sum_{p=1}^P (t_{pi^*} - y_{pi^*})^2 \geq 2E/I \quad (2.19)$$

The new evolved weights are given as $\tilde{w}_{i^*j} = w_{i^*j} + \Delta w_{i^*j}$, for each hidden node $j \in [1, J]$. This results in the new output \tilde{y}_{pi^*} (Equation (2.20)), for the p th input pattern.

$$\tilde{y}_{pi^*} = \Phi \left(\sum_{j=1}^J (w_{i^*j} + \Delta w_{i^*j}) h_{pj} \right) \quad (2.20)$$

$$\sum_{j=1}^J \Delta w_{i^*j} h_{pj} = \Phi^{-1}(\tilde{y}_{pi^*}) - \Phi^{-1}(y_{pi^*}) \quad (2.21)$$

Equation (2.20) can be written as Equation (2.21). This can be simplified by setting $\varepsilon_{pi^*} = \Phi^{-1}(\tilde{y}_{pi^*}) - \Phi^{-1}(y_{pi^*})$. \tilde{y}_{pi^*} is supposed to be closer to the target

t_{pi^*} than y_{pi^*} was, and can thus be defined as $\tilde{y}_{pi^*} = y_{pi^*} + (t_{pi^*} - y_{pi^*}) \cdot \alpha$, where $\alpha \in (0, 1)$. This, together with Equation (2.17), results in Equation (2.22), and thus Equation (2.23).

$$\begin{bmatrix} h_{11} & h_{12} & \dots & h_{1J} \\ h_{21} & h_{22} & \dots & h_{2J} \\ \vdots & \vdots & \ddots & \vdots \\ h_{P1} & h_{P2} & \dots & h_{PJ} \end{bmatrix} \begin{bmatrix} \Delta w_{i^*1} \\ \Delta w_{i^*2} \\ \vdots \\ \Delta w_{i^*J} \end{bmatrix} = \begin{bmatrix} \varepsilon_{i^*} \\ \varepsilon_{2i^*} \\ \vdots \\ \varepsilon_{Pi^*} \end{bmatrix} \quad (2.22)$$

$$\begin{bmatrix} \Delta w_{i^*1} \\ \Delta w_{i^*2} \\ \vdots \\ \Delta w_{i^*J} \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & \dots & h_{1J} \\ h_{21} & h_{22} & \dots & h_{2J} \\ \vdots & \vdots & \ddots & \vdots \\ h_{P1} & h_{P2} & \dots & h_{PJ} \end{bmatrix}^{-1} \begin{bmatrix} \varepsilon_{1i^*} \\ \varepsilon_{2i^*} \\ \vdots \\ \varepsilon_{Pi^*} \end{bmatrix} \quad (2.23)$$

In order to find $\{\Delta w_{i^*j}\}$, the matrix $\{h_{pj}\}$ must be inverted (Equation (2.23)). The complexity of this operation can be reduced by instead approximate the pseudo-inverse, using the least square error criteria as specified by Ben-Israel and Greville (1980). Then, the weight change can be described as in Equation (2.24).

$$\Delta w_{i^*j} = \frac{\sum_{p=1}^P \varepsilon_{pi^*} \cdot h_{pj}}{\sum_{p=1}^P h_{pj}^{-2}} \quad (2.24)$$

Equation (2.24) can be used to calculate the matrix $\{\Delta w_{i^*j}\}$ for each hidden neuron j connected to the output node in question, i^* . Thus we can update the weights, and calculate the error. The error would be the least of the system errors $E_{new}^{(j)}$ that are generated from all the perturbations (Equation (2.25)).

$$\begin{aligned} E_{new}^{(j)} &= \min \left\{ E_{new}^{(j)} \right\} \text{ for } j \in [1, J] \\ &= \frac{1}{2} \sum_{p=1}^P \left(t_{pi^*} - \tilde{y}_{pi^*}^{(j)} \right)^2 \end{aligned} \quad (2.25)$$

In Equation (2.25), $\tilde{y}_{new}^{(j)}$ is the output of the network given the weight change Δw_{i^*j} . The hidden node j that minimizes $E_{new}^{(j)}$ is chosen, and the weights between j and i^* will replace the existing weights. This will thus improve the network, giving better results.

Weight evolution in the inner link While the weight evolution in the outer link is to improve the network, the weight evolution in the inner link is to help the network escape from a local minimum. That is, when the error does not change ($\Delta E = 0$) even though the current error function is above zero ($E > 0$).

Assume an input pattern $p^* \in [1, P]$, such that Equation (2.26) is valid.

$$E_{p^*} = \sum_{i=1}^I (t_{p^*i} - y_{p^*i})^2 \geq 2E/P \quad (2.26)$$

When in a local minima, the weights v_{jk} are evolved such that $\tilde{h}_{pj} = h_{pj}$ for $p = p^*$. The error difference can thus be written as in Equation (2.27), where \tilde{y}_{p^*i} is as in Equation (2.28).

$$\begin{aligned} E - E_{new} &= \frac{1}{2} \sum_{p=1}^P \sum_{i=1}^I \left[(t_{pi} - y_{pi})^2 - (t_{pi} - \tilde{y}_{pi})^2 \right] \\ &= \frac{1}{2} \sum_{i=1}^I \left[(t_{p^*i} - y_{p^*i})^2 - (t_{p^*i} - \tilde{y}_{p^*i})^2 \right] \\ &\geq E/P - \frac{1}{2} \sum_{i=1}^I (t_{p^*i} - \tilde{y}_{p^*i})^2 \end{aligned} \quad (2.27)$$

$$\tilde{y}_{p^*i} = \Phi \left(\Phi^{-1}(y_{p^*i}) + \sum_{j=1}^J w_{ij} \Delta h_{p^*j} \right) \quad (2.28)$$

If the network is to improve, the new system error must be smaller than the last, i.e. $E - E_{new} > 0$. A reduction factor $\lambda \in (0, 1)$ is introduced, as given in Equation (2.29).

$$\frac{1}{2} \sum_{i=1}^I (t_{p^*i} - \tilde{y}_{p^*i})^2 \leq \lambda E/P \quad (2.29)$$

For each output node i , the error is set to be bounded by the average (Equation (2.30)). Here, $t_{p^*i} = \{0, 1\}$, while $y_{p^*i} \in (0, 1)$. To achieve convergence, $|t_{p^*i} - \tilde{y}_{p^*i}|$ ought to be below 0.5. Thus, λ ought to be in the range $(0, \frac{1}{4})$.

$$|t_{p^*i} - \tilde{y}_{p^*i}| \leq \sqrt{2\lambda E/PI} \quad (2.30)$$

Only one of the hidden nodes is changed, to not make things too complex. The node to change, j^* , is a hidden node such that $h_{p^*j^*}$ is near one of the two tails of sigmoidal output (Equation (2.31) and Equation (2.32)). Combined with Equation (2.28), this results in Equation (2.33).

$$\min(h_{p^*j^*}, 1 - h_{p^*j^*}) \leq \min(h_{p^*j}, 1 - h_{p^*j}) \text{ for } \forall j \quad (2.31)$$

$$\Delta h_{p^*j} = \begin{cases} \Delta h_{p^*j^*} & j = j^* \\ 0 & \text{otherwise} \end{cases} \quad (2.32)$$

$$\tilde{y}_{p^*i} = \Phi \left(\Phi^{-1}(y_{p^*i}) + w_{i^*j} \Delta h_{p^*j^*} \right) \quad (2.33)$$

Ng and Leung (2000) then calculates the change in the output of the hidden layer to be as in Equation (2.34). This reveals Equation (2.36), which can be simplified by substituting $\Phi^{-1}(\tilde{h}_{p^*j}) - \Phi^{-1}(h_{p^*j})$ with $\bar{\varepsilon}_{p^*j}$. That gives the matrix in Equation (2.37), where $\bar{\varepsilon}_{p^*j} = 0$ for $p \neq p^*$.

$$\Delta h_{p^*j^*} = \frac{1}{w_{ij^*}} (\Phi^{-1}(\tilde{y}_{p^*i}) - \Phi^{-1}(y_{p^*i})) \quad (2.34)$$

$$\begin{aligned} \tilde{h}_{p^*j^*} &= h_{p^*j^*} + \Delta h_{p^*j^*} \\ &= \Phi \left(\sum_{k=1}^K (v_{j^*k} + \Delta v_{j^*k}) x_{p^*k} \right) \\ &= \Phi \left(\Phi^{-1}(h_{p^*j^*}) + \sum_{k=1}^K \Delta v_{j^*k} \cdot x_{p^*k} \right) \end{aligned} \quad (2.35)$$

$$\sum_{k=1}^K \Delta v_{j^*k} = \Phi^{-1}(\tilde{h}_{p^*j^*}) - \Phi^{-1}(h_{p^*j^*}) \quad (2.36)$$

$$\begin{bmatrix} \bar{x}_{11} & \bar{x}_{12} & \dots & \bar{x}_{1K} \\ \bar{x}_{21} & \bar{x}_{22} & \dots & \bar{x}_{2K} \\ \vdots & \vdots & \ddots & \vdots \\ \bar{x}_{P1} & \bar{x}_{P2} & \dots & \bar{x}_{PK} \end{bmatrix} \begin{bmatrix} \Delta v_{j^*1} \\ \Delta v_{j^*2} \\ \vdots \\ \Delta v_{j^*K} \end{bmatrix} = \begin{bmatrix} \bar{\varepsilon}_{1j^*} \\ \bar{\varepsilon}_{2j^*} \\ \vdots \\ \bar{\varepsilon}_{Pj^*} \end{bmatrix} \quad (2.37)$$

To solve $\{\Delta v_{j^*k}\}$, the input matrix must be reversed, similar to Equation (2.23). Also this matrix can be approximated using the pseudo-inverse with the least square error criteria (Ben-Israel and Greville, 1980), revealing the weight change in Equation (2.38).

$$\Delta v_{j^*k} = \frac{\bar{\varepsilon}_{p^*j^*} \cdot x_{p^*k}}{\sum_{p=1}^P x_{pk}^2} \text{ for } k = 1, \dots, K \quad (2.38)$$

This weight evolution is to be used when trapped in a local minimum. If this happens, it is possible that the matrix in Equation (2.37) is singular. In that case, it must be made non-singular by perturbing the weights connected to a particular hidden node j^* , in order to escape the minimum. After evolving the weights of j^* in the inner link, the weights going out of j^* can be adapted by evolving the outer link as well.

2.2.3.3 Algorithm

The weight evolution algorithm can be summarized as follows:

Step 0: Initialization Initialize the network weights with random values between -0.3 and 0.3.

Step 1: Forward pass Calculate the system error $E(n)$ for epoch n using Equation (2.18).

Step 2: Tolerance test If $E(n) < tolerance$, the solution is good enough, and the learning can *stop*.

Step 3: Error Compute the *error-gradient* $\Delta E = [E(n) - E(n - \tau)] / \tau$, where τ is the window size of the error. That is, the error is considered as a whole throughout the window, such that low error spikes are not sufficient to break learning. The window makes sure that the error is both low and stable.

Step 4: Inner weight evolution If the error is still too high and it is decreasing too slowly — $|\Delta E| < gradient_threshold$ and $E(n) > error_threshold$ — start weight evolution on the inner link, using Equation (2.38).

Step 5: Outer weight evolution For any output node i^* whose error is greater than the average error, start weight evolution on the outer link, using Equation (2.24). Compute the system error of each offspring j using Equation (2.25). Choose the best j^* , i.e. the j^* with the minimum error $E_{new}^{(j)}$, such that $E_{new}^{(j^*)} \leq E_{new}^{(j)}$ for $j = 1, \dots, J$.

Step 6: Backward pass Do backward pass of back-propagation.

Step 7: Continue Go to step 1, unless the error is small enough.

The algorithm includes four parameters that need to be adjusted properly. Ng and Leung (2000) recommend a window size of $\tau = 10$, $gradient_threshold = 0.0001$, $error_threshold = 10 \cdot tolerance$, and $tolerance = 0.001$.

2.2.4 Back-Propagation/Genetic Algorithm

Lu and Shi (2000) introduced a hybrid between the BP and GA, called back-propagation/genetic algorithm (BP/GA). First, BP is used to train the network until a relatively low error is reached. Then, the weights are converted to a bit vector, and a GA is used to evolve the optimal solution.

The hybridization is supposed to give the system a great advantage compared to the pure methods. Lu and Shi (2000) claim that BP/GA is better at converging than both BP and GA, because BP/GA profits from the advantages from both, while each method somewhat removes many of the disadvantages of the other method. More specifically, BP is used to reduce the convergence time of the GA, while the GA saves the network from the local minimum the BP might have caused.

2.2.4.1 Algorithm

A more detailed algorithm can be found in Lu and Shi (2000), but it can be summarized as follows:

Step 0: Initialization Initialize the network weights with random values between -0.3 and 0.3.

Step 1: Back-propagation Use BP to train the network, until the sum squared error $E = \frac{1}{2} |T - Y|^2$ (SSE) either reaches a relatively small value E_0 , or is caught in a local minimum $E_{loc.min}$.

Step 2: Zoom Zoom all the weights by $b = 2^u/w$, where w is the largest of the weights, and $u \in I$. That is, $W_b = bW_{BP}$. Then, reduce the gain factor β to β/b , to stabilize the sigmoid function.

Step 3: Create GA individuals Encode W_b to a binary string G_1 with precision 2^{-v} , $v \in I$, such that each weight is represented as a signed binary code with $u + v + 1$ bits. G_1 forms the initial population of the GA, together with randomly generated genes.

Step 4: Genetic algorithm Run the GA until the globally optimal string G is found.

2.2.5 Genetic Algorithm/Back-Propagation

The opposite of BP/GA has also been tried; Huang et al. (2008) created a genetic algorithm/back-propagation (GA/BP) algorithm to predict highway freight. The difference between BP/GA and GA/BP is mainly the order in which the sub-algorithms are called; GA/BP starts with the GA. That is, it uses GA to decrease the search space, and then uses BP to perfect the weights.

2.2.5.1 Algorithm

The algorithm has the following main steps:

Step 0: Initialization Create child population.

Step 1: Genetic algorithm Run the GA, with fitness assessment, selection and reproduction, until the error is small enough.

Step 2: Back-propagation Use BP to train the network further, until the SSE either reaches a small enough value E_0 , or is caught in a local minimum $E_{loc.min}$.

The details on the sub-algorithms can be found in Section 2.2.1 and Section 2.2.2.

2.2.6 Other Hybrids

There are several other hybrid algorithms for neural networks, used for various domains. This section describes a few of them, both hybrids of BP and GA, and of other algorithms.

2.2.6.1 Hybrid of Differential Evolution and Conjugate Gradients

Bandurski and Kwedlo (2010) introduced two hybrid algorithms, combining differential evolution (DE) and conjugate gradients (CG).

DE is a heuristic algorithm for global optimization over continuous spaces (Storn and Price, 1997). It is an evolutionary algorithm, using vector differences to alter the vector population. According to Ilonen et al. (2003), this method is likely to find the global minimum. However, it would take an intolerable long time.

CG (Charalambous, 1992; Fletcher and Reeves, 1964) is a gradient-based local search technique. It can be used to train ANNs, as a replacement for the steepest gradient descent algorithm.

Lamarckian hybrid of DE and CG: approach 1 (LH1-DECG) uses mainly DE, but uses CG to fine-tune the offspring before they are to compete with their parents. Lamarckian hybrid of DE and CG: approach 2 (LH2-DECG) tries to solve some of the disadvantages with LH1-DECG, by improving both the offspring and the parents with CG.

2.2.6.2 Hybrid of Genetic Algorithm and Simulated Annealing

Abraham and Nath (2000) introduced a hybrid between a GA and simulated annealing (SA) to design an ANN.

SA is a global optimization algorithm that exploits the analogy between the annealing process in metal, and the search for a minimum in a general system. SA has an impressive ability to avoid becoming trapped in local minima. However, it does not usually converge to the global optimum, only a near optimum.

Genetic annealing algorithm (GAA) combines the convergence properties of SA with the parallelization capability of GA. It works as a GA, where each genotype has assigned an energy threshold. This threshold is what determines the fitness; if the threshold of an offspring is less than or equal to the threshold of its parent, the offspring replaces the parent. When every member has been mutated, the population is reheated by changing the threshold.

Abraham and Nath (2000) have designed three hybrid algorithms based on GAA. However, none of these were implemented at the time the paper was written, and the current status is unknown.

The first algorithm is the hybrid algorithm for global search of connection weights. It uses GAA until a minimal required error is achieved, and then BP is used for fine-tuning. This is less sensitive to the initial weight settings than gradient based techniques usually are.

The second algorithm is the hybrid algorithm for global search of optimal architecture. It uses indirect encoding to improve on scalability.

The third algorithm is the hybrid algorithm for global search of learning rules. Here, learning rules are developed. The learning rules are depending on coefficients that will be determined during the global search.

2.2.6.3 Knowledge-Based Hybrid Back-Propagation-Grammatical Evolution Neural Network Algorithm

Turner et al. (2010) introduced the analysis tool for heritable and environmental network associations (ATHENA) as a tool for modeling gene-gene interactions that influence human traits. After adding several changes to ATHENA, they analysed the results.

Their findings indicated that adding a tree-based crossover modification will increase ATHENA's sensitivity for discovering gene-gene interactions.

They also incorporated domain knowledge on gene-gene interactions, which led to a large performance increase. This was specifically advantageous when working with a search space larger than the search coverage. The use of BP to find the network weights would also give a highly statistically significant performance increase.

2.2.6.4 BP/GA Hybrid Method

Yin et al. (2011) introduced a hybrid between a BP algorithm and a GA. The purpose of this algorithm is to optimize the process parameters during plastic injection molding (PIM). The network shall obtain the mathematical relationship between the process parameters and the optimization goals. The process parameters include the mold temperature, melt temperature, packing pressure, packing time, and cooling time. The optimization goals is warpage and clamp force. Their network has two hidden layers, both with 9 nodes. The process parameters gives 5 input nodes, while there are two output nodes; one for warpage and one for clamp force.

Similar work have been done by e.g. Kurtaran et al. (2005, 2006); Shen et al. (2007); Deng et al. (2008, 2010); Zhang et al. (2009); Gao and Wang (2008, 2009); Altan (2010). In contrast to these, Yin et al. (2011) have also taken other factors into consideration, e.g. energy consumption and the production cycle.

The paper presents a multi-objective mathematical optimization model, and a BP/GA hybrid optimization method of PIM process parameters. This is based on the finite element analysis software Moldflow, the Orthogonal experiment method, BP, and a GA. The algorithm gave good results, solving their problem satisfactorily.

2.2.6.5 Hybrid GA Wavelet-BP Algorithm

Su et al. (2009) described a hybrid GA/BP algorithm specialized to handle the highly nonlinear problem of solar radiation prediction. They introduced a wavelet, i.e. a fast-decaying oscillation, to decompose the radiation signal into high and low frequency hefts. The hefts are input to the BP, and better prediction precision is obtained due to the BP's fault-tolerance and nonlinear mapping ability. The GA is there to optimize the weights and threshold values of the BP network. The flow chart of the algorithm can be seen in Figure 2.2 on the next page.

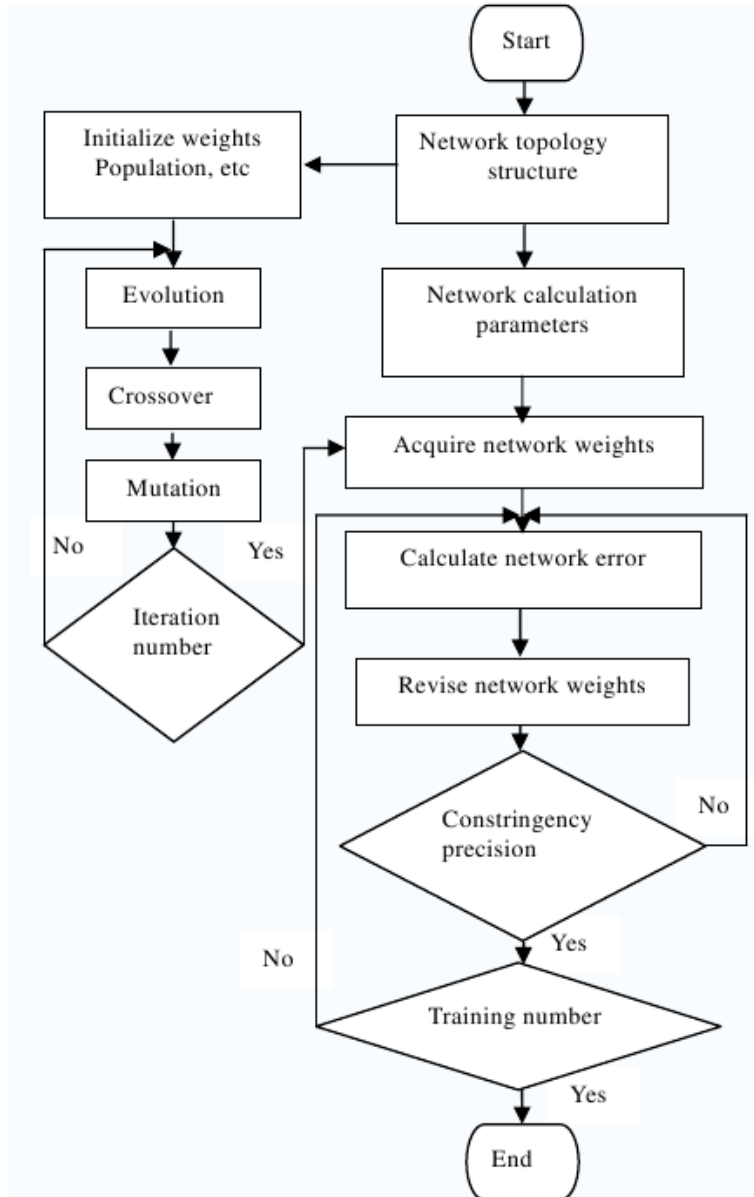


Figure 2.2: Flow chart of the GA wavelet-BP algorithm.

2.2.6.6 Hybrid BP-ANN/GA

Cao and Jin (2007) introduced a hybrid BP-ANN/GA algorithm for classification of urban terrain surfaces. That is, given an image of an area, the algorithm shall classify the different areas as either water, grass, building, road, or flat field. They apply the GA to optimize the initial weights of the BP-ANN, and then perfect the weights using BP. The ANN they use has three layers, with 3 nodes in the input layer, 12 nodes in the hidden layer, and 5 in the output layer — one for each type of terrain.

They compare three algorithms, varying both the learning algorithm and the preprocessing. That is, (1) BP-ANN training with infrared (IR) data, (2) BP-ANN/GA training with IR data, and (3) BP-ANN/GA training with the fusion of IR and synthetic aperture radar (SAR) images.

In the first algorithm, the ANN weights are found using BP only. This would thus correspond to the BP algorithm from Section 2.2.2. In this case, the classification accuracy ranges from 60% (buildings) to 85% (water).

In the second algorithm, the weights are found by first running 200 generations of a GA, and then train the winning set of weights using a BP. This would correspond to the GA/BP algorithm from Section 2.2.5, only with several more generations. In this case, the success rate range from 87% (roads) to 94% (grass). This is considerably better than in the first algorithm. The use of GA accelerated the training speed of the ANN, reducing the number of iterations from 10610 for BP-ANN to only 3424 for BP-ANN/GA. In addition, the computing time is reduced. The BP-ANN use 542 seconds to get below an error threshold of 0.25, while BP-ANN/GA only needed 445 seconds, of which 431 were used by the BP training.

The third algorithm, using the IR/SAR fusion, gave even better results; the accuracy ranged from 92% (flat field) to 95% (building). However, as IR and SAR is special to that domain, the important difference is between algorithm 1 and 2; the comparison of BP-ANN and BP-ANN/GA, given the same preprocessing.

To conclude, Cao and Jin (2007) indicated that BP-ANN/GA is far better than the simpler BP-ANN, as the GA will both optimize the weights, and overcome the slow convergence of the BP-ANN, in addition to help avoid local minima. At least for this domain, and such small networks, a hybrid algorithm of GA and BP is superior to a pure BP algorithm.

2.2.6.7 GA with BP Operator

Osman et al. (2010) introduced the GA/BP hybrid neural network enzyme classification (NNEC). NNEC is a classification algorithm that classifies the class or family of an unknown protein sequence. The protein sequence is represented by a list of numbers, where each amino acid in the protein has its own number, according to the hydrophobicity scale of Kyte and Doolittle (1982). In that scale, each amino acid receives a number ranging from -4.0 to +4.5, depending on the hydrophobicity of the acid. This list is the input given to the network.

The NNEC algorithm is merging GA and BP, with BP being used as an operator

within the GA. That is, it follows the standard genetic algorithm (Section 2.2.1) for the most part. However, after mutation the offspring are improved by running them through 10 or so iterations of BP. This algorithm would thus be the opposite of WE (Section 2.2.3), which is BP with evolution incorporated into each iteration; this is evolution with BP incorporated into each generation.

The algorithm is tested on two network topology profiles; one with between 7 and 10 hidden nodes, and one with between 1 and 5 hidden nodes. In both topologies there are 40 input nodes and 6 output nodes. The profile with most hidden nodes achieved better results than the smaller topology, i.e. 73% accuracy versus 69%. Unfortunately, they have yet to compare it to other classification algorithms. It is thus difficult to predict how it performs compared to a standard GA, both with respect to success rate and computing time.

2.2.7 Tricks and Details to Improve Algorithms

Bullinaria (2009) proposed the use of cross-entropy error function and the softmax activation function for multi-class classification problems. According to Dunne and Campbell (1997), there is a natural pairing between these two functions, and they should be used together.

BPs can be divided into two main types: batch BP and online BP. The algorithm in Section 2.2.2 describes the online BP, where the weights are updated continuously. Batch BP updates the weights only after all the patterns have passed through the network.

2.2.7.1 Softmax

The softmax activation function is only for use in the output layer (Wikibooks, 2010). It is specifically designed for the output nodes, where each node is representing one class. Softmax is giving each of these classes (or nodes) a value, which is the probability that the input pattern belongs to this class (Bishop, 1996). It thus ensures that all the output values are in the range $[0, 1]$, and that all the output nodes sum up to 1.

$$p_i = \frac{e^{q_i}}{\sum_{j=1}^n e^{q_j}} \quad (2.39)$$

The softmax function is represented by Equation (2.39), where p_i is the value of output node i , q_i is the net input to the output node i , and n is the number of output nodes (Bishop, 1996).

2.2.7.2 Cross-Entropy Error Function

The cross-entropy error function (Equation (2.40)) is the output error function. It is used to specify if the error is low enough to stop learning (Bullinaria, 2009; Bishop, 1996), and is used together with the delta function in Equation (2.41), instead of the more common delta function in Equation (2.9) (Bullinaria, 2009).

$$Error = -(t_i^\mu \log(y_i^\mu) + (1 - t_i^\mu) \log(1 - y_i^\mu)) \quad (2.40)$$

$$\delta_i^\mu = t_i^\mu - y_i^\mu \quad (2.41)$$

2.2.7.3 Batch Back-Propagation

As the BP step in the WE algorithm (Section 2.2.3.3) is done after all the patterns have gone through the forward pass, it is clearly done in batch mode. The difference between batch mode and online mode is when the BP occurs; in online mode, the BP is done just after the forward pass, such that each pattern goes through both a forward and a backward pass. This results in continuous change in the network. In batch mode, however, the weight changes are accumulated after each forward pass, and not performed until all the training patterns have been run through the network (Floreano and Mattiussi, 2008; Duda et al., 2000).

An algorithm for batch BP was described by Duda et al. (2000) (Algorithm 2.1).

Algorithm 2.1 Batch back-propagation

```

Initialize network topology.
repeat
  Increment epoch.
  Initialize weight-change matrices  $\Delta w_{ij}$  and  $\Delta v_{jk}$ .
  for all training patterns do
    Push pattern to network.
    Feed-forward.
    Update the weight-change matrices:  $\Delta w_{ij} = \Delta w_{ij} + \eta \delta_{ij} x_i$ ,  $\Delta v_{jk} = \Delta v_{jk} + \eta \delta_{jk} y_j$ 
  end for
  Add the weight change to the weights:  $w_{ij} = w_{ij} + \Delta w_{ij}$ ,  $v_{jk} = v_{jk} + \Delta v_{jk}$ 
until  $Error(w, v) < 0$ 
return  $w, v$ 

```

In each epoch, all the training patterns are iterated over, and the weight changes needed for that specific pattern are computed. However, the weight changes are not added to the network straight away; they are accumulated in the change matrices Δw_{ij} and Δv_{jk} , and are added to the network when all the patterns are finished.

CHAPTER 3

Methodology












The chapter opens with a presentation of the chosen writing systems. Section 3.2 describes the general architecture of the artificial neural network (ANN), while the algorithms investigated in the dissertation are presented in Section 3.3.

3.1 Glyphs and Writing Systems

The writing systems to be classified are chosen based on the classification in Section 2.1. There are two systems from each type, as seen in Table 3.1 on the following page.

The writing systems are compared on a per-pixel basis (Figure 3.1 on page 27). Each writing system is compared to every other writing system, by which each glyph is compared to every other glyph. Two glyphs are compared by iterating over their pixels, summing the difference between the two pixels at the current coordinate. This is summed further, giving each writing system couple a number representing their similarity. This is translated to a grey-scale, where lighter colours represent higher similarity than darker colours. The results (Figure 3.1 on page 27) suggest that the writing systems are not significantly more similar to themselves than to the others. Runic, Hebrew, Thai and Devanagari are among those with the highest self-similarity, while Ugaritic, Hiragana and Han display no significant difference. This complicates the classification process, as the pixels in themselves are insufficient for classification.

Table 3.1: The writing systems to be classified, and their respective type and Unicode values (The Unicode Consortium, 2011). Example glyphs are taken from Ager (2011). The example images are the images used by the ANN.

Type	Script	Code point range	Number of glyphs	Examples	Example image
Alphabet	Runic	U16A0-U16FF	81	ƒꝀƒꝁƒꝂ	
	Ugaritic	U10380-U1039F	30	𐎗 𐎙 𐎛 𐎜 𐎝	
Abjad	Hebrew	U0590-U05FF	30	עברית	
	Arabic	U0600-U06FF	158	العربية	
Syllabary	Canadian Aboriginal Syllabics	U1400-U167F	363	ᓂᓄᓆᓈᓐ	
	Hiragana	U3040-U309F	93	ひらがな	
Abugida	Thai	U0E00-U0E7F	87	ภาษาไทย	
	Devanagari	U0900-U097F	89	देवनागरी लिपि	
Logosyllabary	Han	U4E00-U9FCF	20940	日本語	
	Egyptian Hieroglyphs	U13000-U1342F	1071		

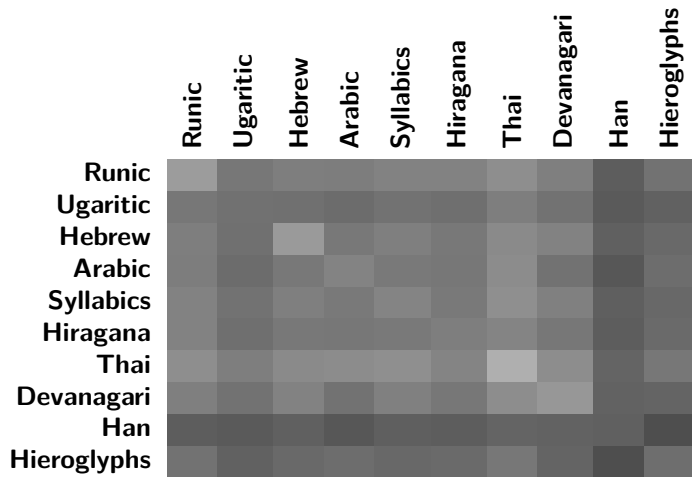


Figure 3.1: All the glyphs in all the writing systems are compared by pixel value. The difference for each pixel is summarized, and the sum is converted to a number between 0 and 1, which is represented as a grey-scale. Lighter colours thus represent higher similarity than darker colours.

3.1.1 Runic

The Runic alphabet was in use throughout Northern Europe: in Scandinavia, the British Isles, and all the way from Germany to the Balkans. The earliest inscriptions found date from the 1st century AD, but most date from the 11th century BC.

The Runic alphabet is also known as *futhark*, after its first six letters. Each glyph is called a *rune*, which in Old Norse means letter or character.

Their place of origin is still unknown. Theories range from it having been modelled on the Latin alphabet, to it being an independent creation, unrelated to other scripts. One thing that is sure, is that the original evolved into other alphabets. There are many different dialects, and they were used in different times and different locations. For example, Norway, Sweden and Denmark developed their own, slightly differing alphabets. The original was probably Elder Futhark, which is thought to be the oldest.

The varying direction of writing and the lack of consistent word division, complicate the reading of runes. However, one eventually settled on writing from left to right, and sometimes dots were used to separate the words. (Ager, 2011)

3.1.2 Ugaritic

Ugaritic cuneiform arose during the 14th century BC in the city of Ugarit, in northern Syria. It was used to write the Ugaritic language, which was closely related to Phoenician. The city of Ugarit flourished from 1400 BC and forth, until

it was destroyed around 1180 or 1170 BC. It was not rediscovered until 1928, when a peasant found an ancient tomb near Ras Shamrah in Syria. The excavation began in 1929, by a group of French archaeologists led by Claude F. A. Schaeffer. (Ager, 2011)

The writing system consists of wedges, making it easy to write in clay. It was usually written from left to right, but not exclusively. The words were divided using a small wedge, but there were no other forms of punctuation. (Ager, 2011; The Unicode Consortium, 2009, Ch. 14)

According to The Unicode Consortium (2011), Ugaritic is an alphabet, while Ager (2011) classifies it as an abjad. As with many of the writing systems, it is difficult to find a good classification because of features from different types. In case of disagreement, this dissertation will use the classification of The Unicode Consortium (2011).

3.1.3 Hebrew

Hebrew is another script originally derived from Phoenician. It was in use as early as the 11th century BC, and is still in use today. It is — together with Arabic — the official language in Israel, with 5 million speakers. Another 2 or 3 million users can be found all over the world.

However, Hebrew was well on its way to extinction; already in 586 BC Aramaic was beginning to take its place, and in 70 AD Hebrew was used mainly for literary and religious function, not as a daily language. This continued until the mid-19th century. Then, Eliezer Ben Yehuda (1858-1922) started to encourage the use of Hebrew, e.g. in his home and at school. Looking at the use of Hebrew today, his approach seems to have worked.

The Hebrew writing system is not used solely for the Hebrew language; it is also used to write Judeo-Arabic, Ladino, Yiddish, and other Jewish languages.

Hebrew is an abjad, written from right to left. It consists of consonants, long vowels, final letters, and diacritics to mark the vowels. The short vowels are normally not marked; the exception is in poetry and in the Bible, as well as material for new learners, like children and foreigners. (Ager, 2011)

3.1.4 Arabic

The Arabic script evolved from the Nabatean Aramaic script, probably around the 4th century AD. The earliest document found dates from 512 AD.

There are two main types of the Arabic script: Classical Arabic and Modern Standard Arabic. Classical Arabic is the language of the Qur'an and other classical literature. Its style and vocabulary differ some from the Modern Standard Arabic, which is the type in use today, in most of written material and formal TV shows. In addition, each region or country has their own dialect, for use in speaking, poetry, cartoons, plays, and personal letters. Arabic is mostly used in the Middle East and the northern parts of Africa. It is used to write e.g. Arabic, Kurdish, Persian, Punjabi, Sindhi, Turkish, and Urdu.

Arabic is written from right to left; however, the numerals are written left to right. Arabic always joins all letters that can be joined, both in hand-writing and in printing. The only exceptions are in crosswords and signs with vertical writing. Due to this joining, the Arabic glyphs look different depending on where in the word they are situated; the stand-alone glyphs are changed so they can be joined to neighbouring glyphs.

As Arabic is an abjad, most glyphs are consonants. There are three long vowels, and the short vowels are marked using diacritics. However, short vowels are only marked in the Qur'an, and sometimes in other religious texts, in classical poetry, and in books for children and foreigners. Sometimes they are also used to decorate the scripts, e.g. in book titles, letterheads, and nameplates. (Ager, 2011)

3.1.5 Canadian Aboriginal Syllabics

Canadian Aboriginal Syllabics is the Unicode unification of Canadian syllabaries, e.g. Blackfoot, Cree, Naskapi, Inuktitut, Carrier, Ojibwe, and Slavey. The syllabics were created in the 1830s by the linguist James Evans, to write the Algonquian languages. The script was adopted by other Canadian aboriginal groups as well, giving a base set of common glyphs, and multiple language-specific glyphs.

Being a syllabary, each glyph consists of one consonant and one vowel. The shape of the glyph determines the consonant, while the vowel is depending on the rotation of said shape. Typically, *es* are turned down, *is* are turned up, *os* are turned left, and *as* are turned right. (Ager, 2011; The Unicode Consortium, 2009, Ch. 13)

3.1.6 Hiragana

Hiragana is a syllabary used to write phonetic Japanese. Originally, the Japanese used the Chinese characters from Han for writing, the *kanji*. However, there are many kanji, and they are all quite complicated. Writing was thus rather difficult for the uneducated. The Japanese then created Hiragana as a simpler writing system, to be used by women. Originally it was called *onnade*, or *women's hand*. Hiragana is a direct simplification of kanji: each Hiragana glyph is a simplification of either a whole kanji or part thereof. For example, ゃ is a simplification of 也, お is a simplification of 於, and ふ of 不. It seems that the men also appreciated a simplification of kanji, because by the 10th century Hiragana was used by almost everybody.

Today, Hiragana is mainly used for particles, grammatical endings, and to indicate the pronunciation. Everything else can be written with kanji, but as there are far too many kanji for any normal person to learn, Hiragana is often used to spell out complicated words. The ratio between Hiragana and kanji is depending on the audience; in children's books there are mainly Hiragana, while academic papers contain mostly kanji. Foreign words — that do not have a kanji — are written in Katakana, another simplification with a one-to-one correlation with Hiragana.

As opposed to Canadian Aboriginal Syllabics, two Hiragana glyphs sharing either a consonant or a vowel have no obvious similarity. As each Hiragana is a

simplification of a Han glyph with the same pronunciation, there is no correlation between looks and sound. (Ager, 2011; The Unicode Consortium, 2009, Ch. 12)

3.1.7 Thai

Tradition has it that Thai was created in 1283 by King Ramkhamhaeng, probably under influence of the Old Khmer alphabet. It is used to write the language of Thailand, Thai. However, it can also be used to write other languages, e.g. Sanskrit and Pali.

Thai is an abugida consisting of 44 basic consonants. Each consonant has an inherent vowel; *o* if it is in the middle of a word, and *a* if it is at the end. In addition, there are five different tones in the language. These are determined by the class of the consonant, whether the syllable is open or closed, the tone marker, and the length of the vowel. Also, there are no word separators in Thai; only space between sentences. (Ager, 2011; The Unicode Consortium, 2009, Ch. 11)

3.1.8 Devanagari

The Devanagari script is an abugida that descended from the Brahmi script around the 11th century AD. It was created to write Sanskrit, but was later adapted to write multiple other languages as well, e.g. Hindi and Nepali.

Each consonant has an inherent vowel, which can be overridden by diacritics or matras. Vowels can either be written as independent letters, or above, below, before, or after the consonant it belongs to.

The name Devanagari comes from Sanskrit. The word *deva* means *god*, or *celestial*, while *nagari* means *city*. It is uncertain precisely what this means, and why it is named such. (Ager, 2011; The Unicode Consortium, 2009, Ch. 9)

3.1.9 Han

The Han script seems to have originated in China, probably sometime during the second half of the 2nd millennium BC. The earliest recognizable glyphs were inscribed on ox bones and turtle shells, and date from the Shang dynasty, 1500 to 950 BC. The script on these bones is known as 甲骨文 (*jiǎgǔwén*, or *shell bone writing*), and the bones were used for divination. They were heated, and the resulting cracks were inspected to determine the answers to questions, usually regarding hunting, warfare, weather, and the like. This is not the earliest of Chinese writing; Neolithic pottery from 4800 BC have been found, with inscriptions that could be some form of writing. However, these symbols do not resemble the characters found on the shell bones, and are probably not related.

The characters were originally pictures of people, animals and things, not too far from the original cave paintings. However, they have been stylised over the centuries, and are no longer recognizable. The full evolution from the oracle bones to today's script can be seen in Figure 3.2 on the next page. Some of these scripts are still in use today. The Large and Small Seal scripts are used mainly for personal names and company names. The Grass script is a cursive script, used mainly used

	horse	cart	fish	dust	see	
Oracle bone script 甲骨文 (jiǎ gǔ wén)						The Oracle bone script was used during the Shang or Yin Dynasty (c. 1400-1200 BC)
Bronze script 金文 (jīn wén)						The Bronze script was used during the Zhou Dynasty (c. 1100 - 256 BC)
Large Seal script 大篆 (dà zhuàn)						The Large Seal script was used during the Zhou Dynasty (c. 1100 - 256 BC)
Small Seal script 小篆 (xiǎo zhuàn)						The Small Seal script was used during the Qin Dynasty (221-207 BC)
Clerical script 隸書 (lì shū)						The Clerical and Standard scripts first appeared during the Han Dynasty (207 BC - 220 AD).
Standard script 楷書 (kǎi shū)						
Running script 行書 (xíng shū)						The Running script has been used for handwritten Chinese since the Han Dynasty.
Grass script 草書 (cǎo shū)						The Grass script is the Chinese equivalent of shorthand and has been used since the Han Dynasty.
Simplified script 简体字 (jiǎntǐ zì)						The Simplified script has been used in the P.R.C. since 1949. It is also used in Singapore.
hànyǔ pīnyīn 汉语拼音	mǎ	chē	yú	chén	jiàn	<i>Hanyu pinyin</i> has been used in the P.R.C. since 1958.
zhùyīn fúhào 注音符號	ㄇˇ	ㄔㄨㄛ	ㄩˊ	ㄔㄨㄣˊ	ㄐㄧㄢˋ	<i>Zhuyin fuhao</i> was developed in China in 1913 and is still used in Taiwan.

Figure 3.2: The evolution of the Han writing system, from the early oracle bones to Simplified Chinese. Image is from Ager (2011).

for calligraphy. The Simplified script came in the early 1950s, when it was decided to simplify the language to reduce the illiteracy. Today, Simplified Chinese is in use in Singapore and mainland China, while Traditional Chinese is used in Hong Kong, Macao, Taiwan, and in Chinese communities all over the world.

Chinese writing was not only in use in China; both Korea and Japan have developed their writing systems from Chinese. Korea started already during the Chinese occupation from 108 BC to 313 AD. They wrote Classical Chinese by the 5th century AD, and three different writing systems were developed based on Chinese; *Hyangchal*, *Gukyeol*, and *Idu*. *Hyangchal* used the Chinese characters to represent the sounds of Korean, and was mainly used for poetry. *Idu* combined Chinese characters with special symbols, that indicate grammatical endings and the like, and was used for documents. In 1444, the Korean alphabet, *Hangul*, was invented. However, most Koreans continued to write either in Classical Chinese or in Korean using either *Gukyeol* or *Idu*. *Hangul* was associated with low status people, i.e. women, children, and uneducated people. During the 19th and 20th century, a new writing system arose, combining Chinese characters (*hanja*) and *Hangul*. Since 1945, however, the use of *hanja* has decreased.

Japan joined in during the Standard script phase, in the 4th century AD. Then they began to import and adapt aspects of Chinese culture, e.g. the script. Originally, the Japanese wrote in either Classical Chinese or in a Japanese-Chinese hybrid style. Then, they started to use the Chinese characters to write Japanese, using the phonetic values of the characters. Eventually, Chinese characters were used to write both words directly borrowed from Chinese, and Japanese words with similar meaning, creating *kanji*. At the same time, Chinese characters were used for their phonetic values to write grammatical elements. These characters were simplified, and over time they became the syllabaries *Hiragana* and *Katakana*.

Many of the characters have been combined to create new ones. An example is the Chinese word for *computer*, which literally means *electric brain*, 電腦. Each of these characters are also combinations of other characters. It all starts with 天, *sky*. This develops to 雨, *rain*, by adding the smaller rain strokes. Rain over a *field* tend to mean *thunder* (雷). Add a power cord to the thunder, as on an electrical appliance, and there is *electricity* (電). It is also an appropriate character regarding another matter; rain, thunder, and lightning are all forces of nature, and so is electricity. The next main character is *brain*, 腦. This consists of the characters *meat*, *flow*, and *smart*. In other words, a brain is smart meat that thinks. The flow is there to express the *flow of thought*, which perhaps represents the feed-forward through the natural neural network. The result is the electric brain, 電腦.

These smaller glyphs within one character are called *radicals*. About 90% of Chinese characters contain a radical that hints about the meaning, and a phonetic component that hints about the pronunciation.

3.1.10 Egyptian Hieroglyphs

Egyptian Hieroglyphs were used to write Egyptian, the Afro-Asiatic language spoken in Egypt until the 10th century AD. It is still in use as a liturgical language

of the Christians in Egypt, the Copts. However, it is now written using Coptic, not Hieroglyphs.

Egyptian Hieroglyphs are very detailed drawings, and were thus not suitable for everyday writing. Consequently, the Hieratic script evolved as a simplification of Hieroglyphs. Egyptian Hieroglyphs in all their detail were used mainly for formal inscriptions, e.g. on tombs and temples.

The direction of writing concerning Egyptian Hieroglyphs varied; it was written from right to left, left to right, and top to bottom. Since Hieroglyphs were used on monuments and walls, they were mainly decorations, and were thus written to increase the aesthetics of the decorations. For simplicity, the Hieroglyphs turn according to the direction; they always look towards the beginning of the line. (Ager, 2011; Collier and Manley, 2003)

According to the ancient Egyptians, the Hieroglyphs were invented by the god Thoth. The Hieroglyphs were thus called *mdwt ntr*, *God's words*. The word *hieroglyph* was first used by Clement of Alexandria (150-215 BC), and is derived from the Greek words *hieros* (*sacred*) and *glypho* (*inscriptions*) (Ager, 2011; Osborn, 2005).

It is possible that the Hieroglyphs predates the Sumerian Cuneiform writing. If this is the case, it is the oldest known writing system. It is also possible that the two systems developed simultaneously. Either way, the Egyptian Hieroglyphs comprise a very old writing system; the earliest Hieroglyphs are dated as far back as 3400 BC.

It was the Emperor Theodosius I that late in the 4th century AD ordered the closure of all pagan temples throughout the Roman empire. Consequently, the latest dated inscription was carved on the gate post of a temple at Philae in 396 AD. Soon, all knowledge of the Hieroglyphs were lost. That is, until Jean-François Champollion (1790-1832) finally managed to decipher the script, thanks to the Rosetta stone. (Ager, 2011)

3.2 Artificial Neural Network Design

The ANNs used have the same three-layered architecture. There are 400 input nodes plus one bias node, 50 hidden nodes plus one bias, and up to ten output nodes.

3.2.1 Nodes

The number of input nodes is the number of pixels in the glyph images. This must be balanced between enough pixels to separate the glyphs and few enough to have an acceptably low execution time, so the trick is to as low as possible without compromising the image quality too much. By creating glyphs of different resolutions, those around 20 times 20 were found to be a good compromise (Rødland, 2010).

The number of hidden nodes was chosen a bit more randomly. 50 was chosen because it gave an acceptable result in acceptably short time. 100 hidden nodes

added more to the execution time than to the quality of the results.

The number of output nodes, on the other hand, is undoubtedly correct; there is one output node for each writing system to be classified. Each output node has thus its own writing system, and the value of the node describes the probability that the glyph in question is of that specific writing system. This output design is to help analyse the relationship between the writing systems. By using the probability for each of the writing systems, it is easier to pick up on common misclassifications and other patterns, e.g. if it really *is* harder to separate between Han and Hiragana than, say, Han and Thai.

The two bias nodes were chosen to add a threshold to the nodes (Jones, 2009; Floreano and Mattiussi, 2008; Callan, 1999). The bias was originally 1, but experimentation on the values 0, 0.5 and 1 indicated that 0.5 was the best value for both bias nodes.


3.2.2 Architecture

The final architecture, as well as the whole path from Unicode point to output values, can be seen in Figure 3.3 on the facing page.

The starting point is the Unicode value, e.g. #13001. This is converted to a glyph image, which then is resized to a 20×20 image. This results in a list of 400 pixels, each with an integer value between 0 and 255, inclusive. Each of these pixels are run through a sigmoid function (Equation (3.1)), such that the final value is a floating point between 0 and 1, inclusive. In addition to reducing the numbers to a more ANN friendly value, the sigmoid function widens the gap between the light and the dark pixels, creating more of a binary situation. The 400 floating points are then given to one input node each, and the values propagate through the network.

$$\Phi(n) = \frac{1}{1 + e^{-0.1(n-127)}} \quad (3.1)$$

The final output is a normalized list of the values of the output nodes. That is, they sum to 1.0, or 100%. Each of these values is the probability that the glyph in question is of the writing system coupled with that specific output node. A perfect classification of #13001 would thus be (0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 1.000), as the glyph¹ is an Egyptian Hieroglyph, which is coupled to the last output node. The writing systems are coupled with nodes in the order given in Table 3.1 on page 26.

¹The glyph used in Figure 3.3 on the next page is . This is a determinative, or meaning-sign, used to illustrate that the word it follows has something to do with the mouth, either literally or metaphorically. That is, it follows words for what can be taken in or expelled through the mouth. As it is a determinative, the glyph has no sound of its own; it is simply a written construct to give the reader more information and context. (Collier and Manley, 2003)

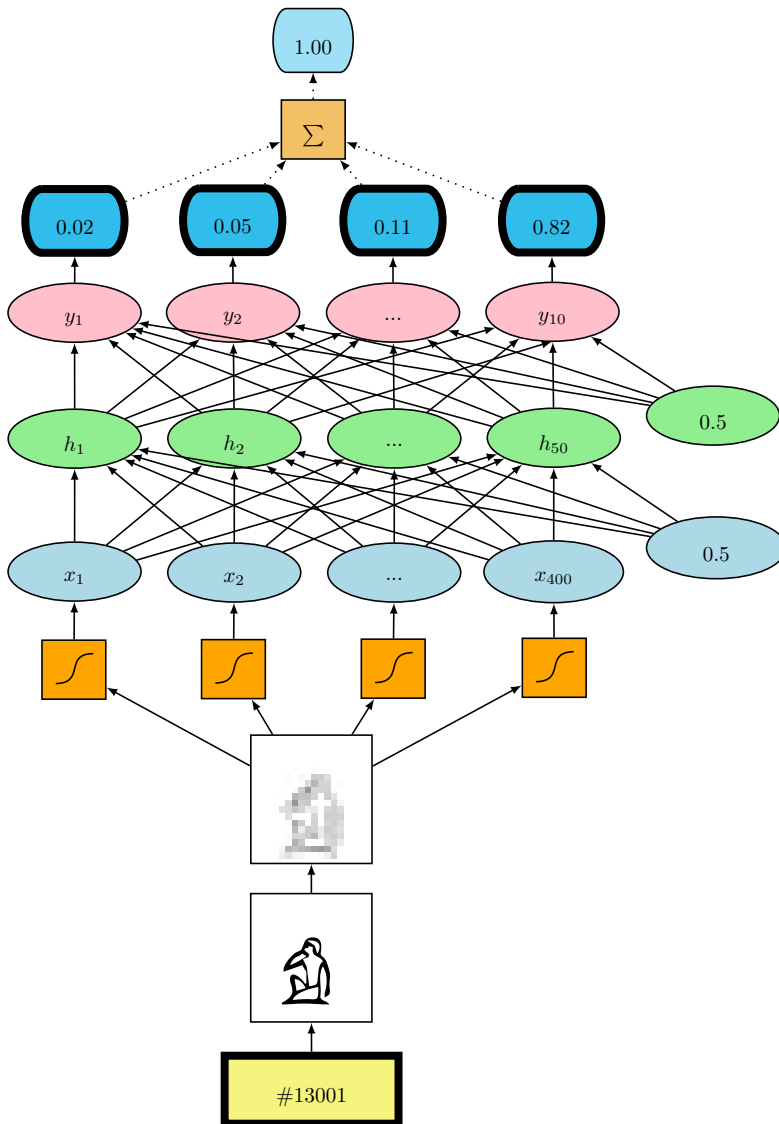


Figure 3.3: The path from glyph to output values is long and complex. The starting point is the Unicode value, i.e. #13001, which is converted to a glyph image. This is resized to a 20×20 image. Each of the 400 pixels are given to an input node in the ANN, after going through a sigmoid function (Equation (3.1)). The output of the ANN is a list of normalized numbers, one for each output node, i.e. one for each writing system. The value is the probability that the glyph in question is of that specific writing system.

Table 3.2: Parameters used by the algorithms.

Parameter	BP	WE	BP/GA	GA/BP
<i>Learning</i>				
Learning rate	0.01	0.01	0.01	0.01
<i>Evolution</i>				
Crossover rate	–	–	0.01	0.01
Mutation rate	–	–	0.001	0.001
Culling	–	–	0.3	0.3
Elitists	–	–	5	5
Max number of children	–	–	100	100
Number of generations	–	–	200	10
Gene length	–	–	20	20
Number of genes	–	–	24461	24461
Population size	–	100	100	100
<i>Local minima and stopping condition</i>				
Tolerance	–	0.01	0.001	0.001
Error threshold	–	0.1	0.01	0.01
Gradient threshold	–	10^{-4}	10^{-10}	10^{-10}
Window size	–	5	10	10

3.3 Algorithms

Four algorithms are used and compared. There are three algorithms that combine learning and evolution in different ways, and a pure back-propagation (BP) algorithm used for comparison.

The basis of all the algorithms is the standard BP algorithm (Floreano and Mattiussi, 2008), but with certain tricks. According to Bullinaria (2009); Dunne and Campbell (1997); Bishop (1996), the softmax activation function and the cross-entropy error function are a good pair for multi-class classification problems. These functions are thus used instead of the standard functions recommended by Floreano and Mattiussi (2008).

The parameters used in the algorithms can be found in Table 3.2. The local minima parameters are lower in the GA algorithms than in WE. After all, WE should be more sensitive to local minima than the other algorithms, as they trigger evolution and thus should be reached multiple times. In the GA algorithms, a local minimum is a reason to stop the algorithm, so it should not catch insignificant local minima. The mutation and crossover parameters are unusually low; these are found by trial and error.

3.3.1 Genetic Algorithm

Genetic algorithms (GAs) (Algorithm 3.1) use a bit vector genotype, both mutation and crossover, full generational replacement, uniform parent selection, and fitness proportionate mate selection. The flowchart in Figure 3.4 displays the main process of the algorithm.

Algorithm 3.1 GA algorithm

Create child population.

repeat

 Fitness assessment.

 Selection.

 Reproduction.

until error \leq tolerated **or** 200 generations

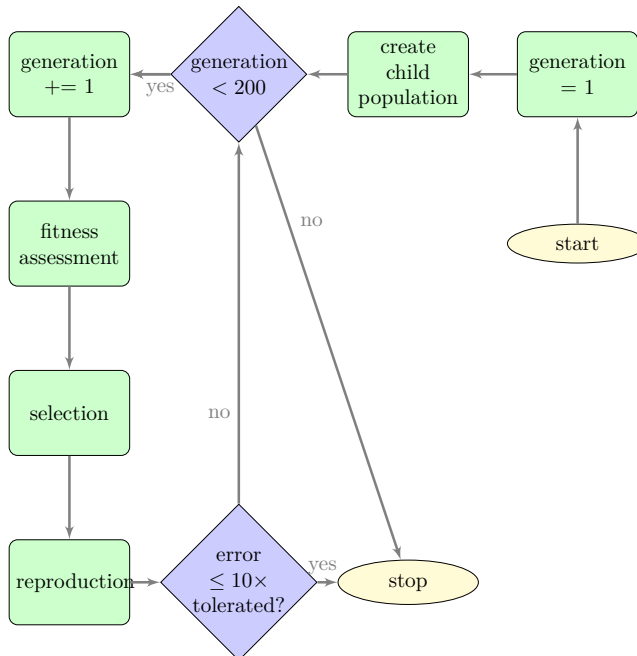


Figure 3.4: Flowchart for GA. The evolution is performed for 200 generations or until the error is less than ten times the tolerated. During the fitness assessment, the ANNs are tested with the training set of glyphs. The best ANNs are selected, and they reproduce, using mutation and/or crossover.

Development As this is a GA, the genotype is a bit vector. Each chromosome consists of one gene for each weight, where each gene consists of 8 bits, configured

Table 3.3: The gene consists of 8 bits, and specifies whether there is a connection (bit 0), the sign of the weight (bit 1) and the weight value (bit 2-7).

0	1	2	3	4	5	6	7
Connection	Sign	10^0	10^{-1}	10^{-2}	10^{-3}	10^{-4}	10^{-5}

as in Table 3.3. The first bit is the connection bit, inspired by EEC (Shi and Wu, 2008). If it is 0, there is no connection; i.e. the weight is automatically set to 0. If 1, the weight is as specified by the later bits. The second bit is the sign. If it is 0, the weight is positive; if 1, the weight is negative. The rest of the bits specify the weight value. The first of these bits specifies the ones, then follows the tenths, the hundreds and so on. This results in a weight range from -1.96875 (11111111) to 1.96875 (10111111).

Fitness Testing The fitness of the individual is depending on the classification results. The weights created during development are pushed on an ANN, and then the ANN is used to classify the glyphs in the training set. The error is calculated as the average error for all the glyphs.

$$TSS = \frac{1}{M} \sum_{\mu} \left(\frac{1}{N} \sum_i (t_i^{\mu} - y_i^{\mu})^2 \right) \quad (3.2)$$

The error function used is not the same as during learning, but the total sum squared error (TSS) (Equation (3.2)).

The fitness of an individual is calculated based on the actual error and the maximal error. As the error function is (3.2) and the possible values for each node is between 0 and 1, the error must necessarily be between 0 and 1. The fitness function is thus:

$$fitness(error) = 1 - error \quad (3.3)$$

Adult Selection As specified by the algorithm (Section 2.2.1), GAs shall have a full generational replacement, where all of the children — and only the children — survive into adulthood. As a result, there is virtually no selection pressure. This gives all the individuals the possibility to stay and maybe develop into a very good individual in some generations. However, this is a slow process and success is in no way guaranteed.

To speed up the process, culling was introduced (Baum et al., 2001; McQuesten and Miikkulainen, 1997). Real-life culling is the process of removing animals from a group based on specific criteria, either to reinforce good characteristics or to remove undesired characteristics (Merriam-Webster, 2010). The same process is implemented here; the individuals are sorted based on fitness, and a certain percentage of the best children are kept, while the rest are killed off. The adult population is thus much smaller than the child population.

Parent Selection There is a uniform parent selection, as specified by the algorithm (Section 2.2.1). It is not, however, as uniform as it was meant to be, as the suboptimal individuals were killed off in the adult selection. Instead of a uniform selection between all the individuals, it is thus a uniform selection between the best individuals.

Reproduction The reproduction follows the algorithmic guidelines given in Section 2.2.1. The parents are chosen from the adult pool with a probability proportional to their fitness. There is a certain chance the parents are mating using crossover, otherwise they are just directly copied into the next generation. In case of crossover, their chromosomes are split on the border between two genes, to prevent the destruction of two potentially good genes. Then, for each gene in each of the new chromosomes, there is a small chance of a mutation, i.e. a bit flip of a random bit.

The newborn children are not the only ones competing for a place in the next generation. There are also two elitists; the very best from the last generation. In addition to having a high probability of giving their genes to the next generation through mating, they are allowed to join in themselves. This is to prevent good individuals from dying off without contributing to even better children; if the next generation is not superior to the last, at least the best individual is not any worse.

Stopping Criteria The evolution was meant to continue until one of the individuals reached the error threshold. However, the evolution was a slow process, with a very slowly decreasing error. The available memory, on the other hand, tended to disappear very quickly. Consequently, the whole process always died of memory fault, long before this error was reached. (Rødland, 2010)

To solve this problem, the evolution continues only until the error has reached 0.1, but for no more than 200 generations. Then, a neural network is created based on the best individual, and that ANN is used for testing.

3.3.2 Back-Propagation

BP (Algorithm 3.2) is the online-learning back-propagation algorithm that is the basis of the others. The learning algorithm is based on the standard BP algorithm on page 9 — as displayed in Figure 3.5 on the next page — but using the tricks in Section 2.2.7. This includes the softmax activation function and the cross-entropy error function.

Weight Initialization All the weights are initialized according to Equation (3.4), where K is the number of nodes in the link's input layer. This gives a number between -0.3 and 0.4, before it is divided on the number of input nodes K . The weights are divided on K to make sure that the input values for the nodes in the next layer will be at an acceptably low level, even given the large number of incoming weights.

Algorithm 3.2 BP algorithm

```

Initialize weights.
repeat
  for all glyphs do
    Feed-forward.
    Back-propagation of error.
  end for
until error ≤ the tolerated

```

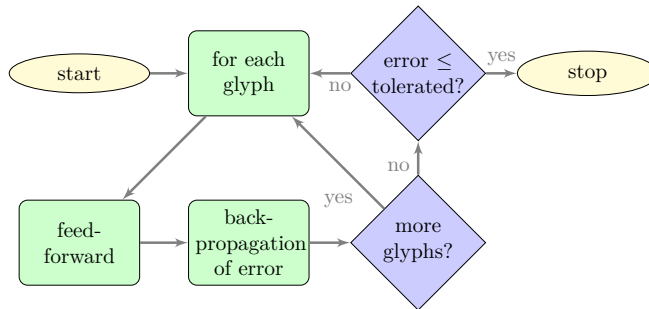


Figure 3.5: Flowchart for BP. For each glyph, the image pattern is input to the network, and is fed forward. Then the weights are changed when back-propagating the error. This continues until the error is low enough.

$$w = \frac{\text{random}(70) - 30}{100K} \quad (3.4)$$

One might think that using 35 instead of 30 — thus initialize to a number between -0.35 and 0.35, with an average of 0 — would be a better solution. However, experimentation showed that this led to a slowly increasing error, instead of a faster decreasing error. Initialization between -0.4 and 0.3 led to a faster increasing error. A number between -0.3 and 0.4 seemed thus like best option.

From Glyph to Input Node For each of the training patterns, the pixels are mapped onto the input nodes. However, the pixel values are not directly copied. They are sent through the sigmoid function in Equation (3.5), giving input values in the range $[0, 1]$.

$$\Phi(n) = \frac{1}{1 + e^{-0.1(n-127)}} \quad (3.5)$$

From Input Node to Output Node The values go from the input layer to the hidden layer through Equation (3.6a), and further to the output layer by using

Equation (3.6b).

$$h_j^\mu = \Phi \left(\sum_k v_{jk} x_k^\mu \right) \quad (3.6a)$$

$$y_i^\mu = \Phi \left(\sum_j w_{ij} h_j^\mu \right) \quad (3.6b)$$

The softmax activation function (3.7) is used as the sigmoid function for the output layer. As this function is specifically designed for multi-class classification problems (which the glyph classification problem is), it is the natural choice. However, as it is only for use in the output layer, Equation (3.8) was used for the hidden layer. (3.8) is a specialized version of Equation (2.6), with $c = 1$.

$$p_i = \frac{e^{q_i}}{\sum_{j=1}^n e^{q_j}} \quad (3.7)$$

$$\Phi(a_i) = \frac{1}{1 + e^{-a_i}} \quad (3.8)$$

Error and Weight Change The errors for the output layer and the hidden layer are calculated using Equation (3.9a) and Equation (3.9b), respectively. Equation (2.41) is used because of the softmax function and the cross-entropy error function.

$$\delta_i^\mu = t_i^\mu - y_i^\mu \quad (3.9a)$$

$$\delta_j^\mu = h_j^\mu (1 - h_j^\mu) \sum_i w_{ij} \delta_i^\mu \quad (3.9b)$$

The weight change (Equation (3.10)) is based on Equation (2.12).

$$\Delta w_{ij}^{\mu t} = \delta_i^\mu h_j^\mu \quad (3.10a)$$

$$\Delta v_{jk}^{\mu t} = \delta_j^\mu x_k^\mu \quad (3.10b)$$

The final weight is then updated as in Equation (3.11), by increasing the weight by the product of the weight changes and the learning rate $\eta = 0.01$.

$$w_{ij}^t = w_{ij}^{t-1} + \eta \Delta w_{ij}^{\mu t} \quad (3.11a)$$

$$w_{jk}^t = w_{jk}^{t-1} + \eta \Delta w_{jk}^{\mu t} \quad (3.11b)$$

Error and Stopping Criteria The final error is the cross-entropy error function in Equation (2.40), as proposed by Bullinaria (2009). The learning continues until the error is less than 0.01 (Equation (3.12)). When the error is this low, the accuracy is acceptably high, and it does not take *too* long to run.

$$Error = |-(t_i^\mu \log(y_i^\mu) + (1 - t_i^\mu) \log(1 - y_i^\mu))| < 0.01 \quad (3.12)$$

3.3.3 Weight Evolution

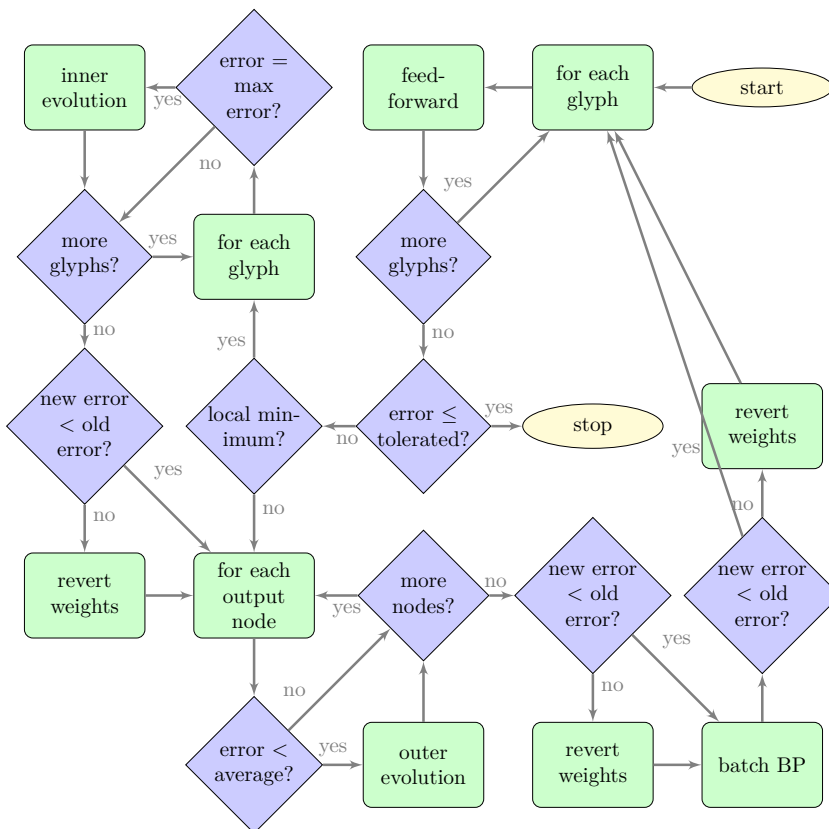


Figure 3.6: Flowchart for WE. First, all the glyphs are going through feed-forward, and the error is accumulated. If the network is stuck in a local minimum, the inner evolution is started. Then, the outer evolution is run, for output nodes with an error above average. Then, the error change is adding in a batch back-propagation, and the circle continues.

Weight evolution (WE) is a merging of learning and evolution (Algorithm 3.3 on the facing page and Figure 3.6). It is inspired by Ng and Leung (2000); however, it is not a reimplementaion. Only the main idea from Ng and Leung (2000) is used, i.e. the concept of evolving the outer and inner weights to improve the performance and emerge from local minima.

It is important to note that this is not an evolutionary algorithm that evolves the weights of a neural network; it is a back-propagation algorithm that includes some aspects of evolution.

This is the only back-propagation algorithm of the four that uses batch learning (Algorithm 2.1 on page 24). After the feed-forward stage, the weights connected

to the output nodes with especially high error are evolved. That is, the weights are copied and each weight is perturbed. The set of weights that reduces the error the most is inserted instead of the original set. If the algorithm is stuck in a local minimum, the same evolution is performed on the weights going into the hidden node with the worst performance. This is only tested with the most problematic patterns. After another feed-forward, batch back-propagation is performed.

The error is recalculated during the evolutionary phases, to assure that the evolutionary change is based on the current network. It also adds more complexity and runtime to the algorithm.

After all types of weight perturbation, the weight change is reverted in case the network as a whole performs worse than it did before the weight change. This little trick makes sure that the network is improved for each iteration, but it also causes a large amount of computation without having any change in the network.

Algorithm 3.3 WE algorithm

```

Initialize weights.
repeat
  for all glyphs do
    Feed-forward.
  end for
  if stuck in a local minimum then
    for particularly problematic glyphs do
      Create multiple copies of the weights between the input nodes and one of
      the hidden nodes, then mutate each weight.
      Choose the set of weights that lowers the error the most.
    end for
    if the total error is larger than before then
      Revert to the original weights.
    end if
  end if
  for the output nodes with most error do
    Create multiple copies of the weights connected to each node, and mutate
    each weight.
    Choose the set of weights that lowers the error the most.
  end for
  if the total error is larger than before then
    Revert to the original weights.
  end if
  Feed-forward.
  Compute error change.
  Back-propagation of error.
  if the total error is larger than before then
    Revert to the original weights.
  end if
until error  $\leq$  the tolerated or 300 epochs
  
```

3.3.4 Back-Propagation/Genetic Algorithm

Back-propagation/genetic algorithm (BP/GA) (Algorithm 3.4 on the facing page and Figure 3.7) starts with the standard BP algorithm to reduce the search space, and then it uses a GA to find the best solution. It is based on the work of Lu and Shi (2000). That is, their algorithm is not reimplemented in all its detail, but the main idea is used as a basis.

In BP/GA, BP is used to reduce the search space, while GA finds the optimal solution in that space. The back-propagation algorithm is first run until it is either stuck in a local minimum, or the error is below two times the tolerated error. Then, the GA is used to find the best solution. The GA continues until the error is low enough but for no more than 200 generations.

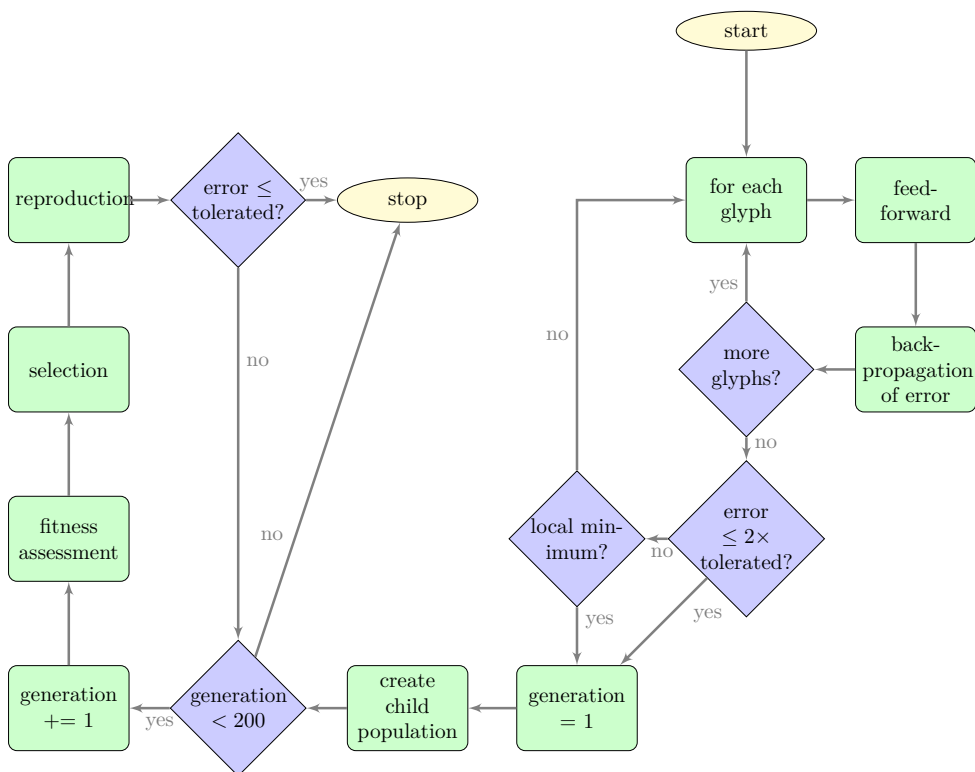


Figure 3.7: Flowchart for BP/GA. First, the BP is run until the error is low enough, or a local minimum encountered. The current network is translated to a bit vector using Table 3.3 on page 38, and an initial population is created based on this original bit vector. This population goes then through the standard evolutionary circle, until either the error is small enough, or until there have been 200 generations.

Algorithm 3.4 BP/GA algorithm

```

Initialize weights.
repeat
  for all glyphs do
    Feed-forward.
    Back-propagation of error.
  end for
until error  $\leq$  2 times the tolerated or local minimum
Translate ANN to bit vector.
Create child population based on bit vector.
repeat
  Fitness assessment.
  Selection.
  Reproduction.
until error  $\leq$  the tolerated or 200 generations

```

3.3.5 Genetic Algorithm/Back-Propagation

Genetic algorithm/back-propagation (GA/BP) (Algorithm 3.5 and Figure 3.8 on the following page) is based on BP/GA. The sub-algorithms are the same; they are simply reversed, such that GA decreases the search space and BP locates the optimal solution. This is not a new idea; Huang et al. (2008) used the same approach to forecast highway freight.

GA/BP runs the GA until the error is ten times the tolerated value but for a maximum of ten generations. Then, BP is used to find the optimal solution, and it keeps going until the error is below the threshold.

Algorithm 3.5 GA/BP algorithm

```

Create child population.
repeat
  Fitness assessment.
  Selection.
  Reproduction.
until error  $\leq$  10 times the tolerated or 10 generations
repeat
  for all glyphs do
    Feed-forward.
    Back-propagation of error.
  end for
until error  $\leq$  the tolerated

```

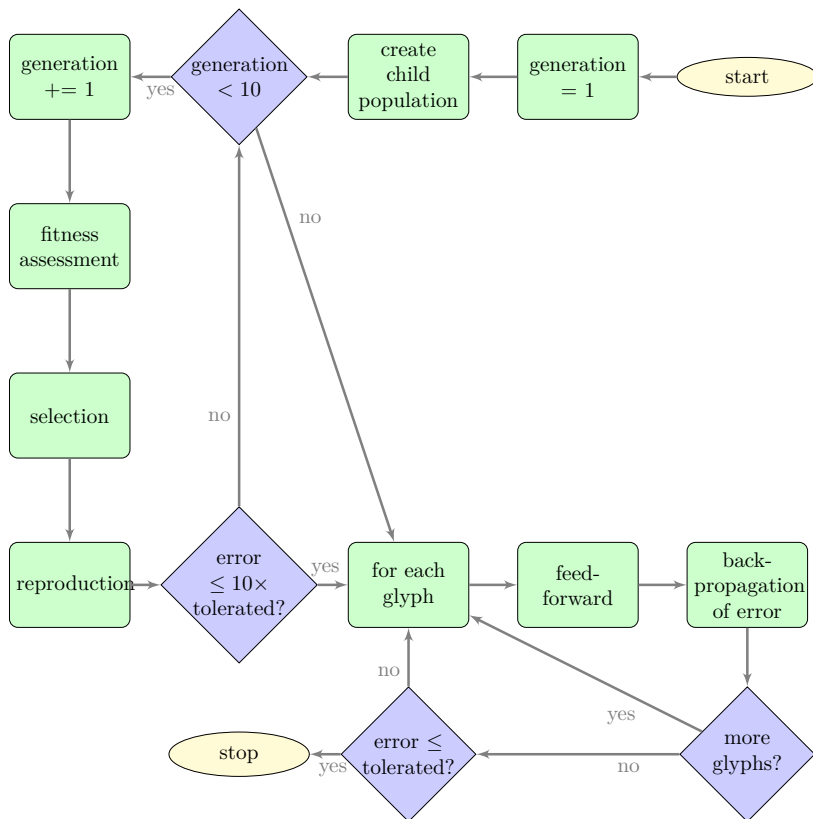


Figure 3.8: Flowchart for GA/BP. A random bit vector is created, and the initial population is created based on this. Evolution is started, and continues for ten generations, or until the error is less than ten times the tolerated. Then, the best individual is improved using back-propagation. BP continues until the error is within the toleration limit.

CHAPTER 4

Results and Discussion

The entire set of result graphs can be found in Appendix B. Section B.1 contains a short introduction to the different kinds of graphs, describing how to read them. The main results are found in Section B.2, sorted by both algorithm and phase. Section B.3 includes graphs based on (1) a maximum of 25 runs, and (2) a maximum of 3 runs. This is to check if the good test results of back-propagation (BP) and genetic algorithm/back-propagation (GA/BP) are due to the vast number of available runs for these two algorithms, in contrast to the few runs of weight evolution (WE) and back-propagation/genetic algorithm (BP/GA). Finally, Section B.4 contains the graphs from a pure evolutionary algorithm, the genetic algorithm (GA) from Rødland (2010).

The most important results are discussed in this chapter. Section 4.1 compares the four algorithms with each other, based on the main results. Section 4.2 compares the writing systems with each other, discussing which features are important to the classification. Finally, the result graphs from Section B.3 are discussed in Section 4.3.

4.1 Algorithm Comparison

In this section, the algorithms are compared to each other based on the graphs found in Section B.2.2.

4.1.1 Training

All the algorithms except the WE algorithm have statistically significant results. There are 120 runs of BP and GA/BP, 25 of BP/GA and 3 runs of WE. These are the results gained after almost four weeks of runs. Even if not all of these results are

very convincing with regard to the decreasing error, they clearly show the difference in runtime. As experienced by Rødland (2010), BP learning is significantly faster than GA evolution. Both BP and GA/BP are — relatively speaking — fast. 10 runs take roughly one day to execute. BP/GA is considerably slower, with only one run per day, even though the GA part is run for a maximum of 200 generations. If it had continued until the error was small enough, as BP and GA/BP do, it would have been even slower. In that case, the algorithm would probably display better results, as the error would have been smaller when the testing started. Due to the runtime, however, it would still be inferior to both BP and GA/BP.

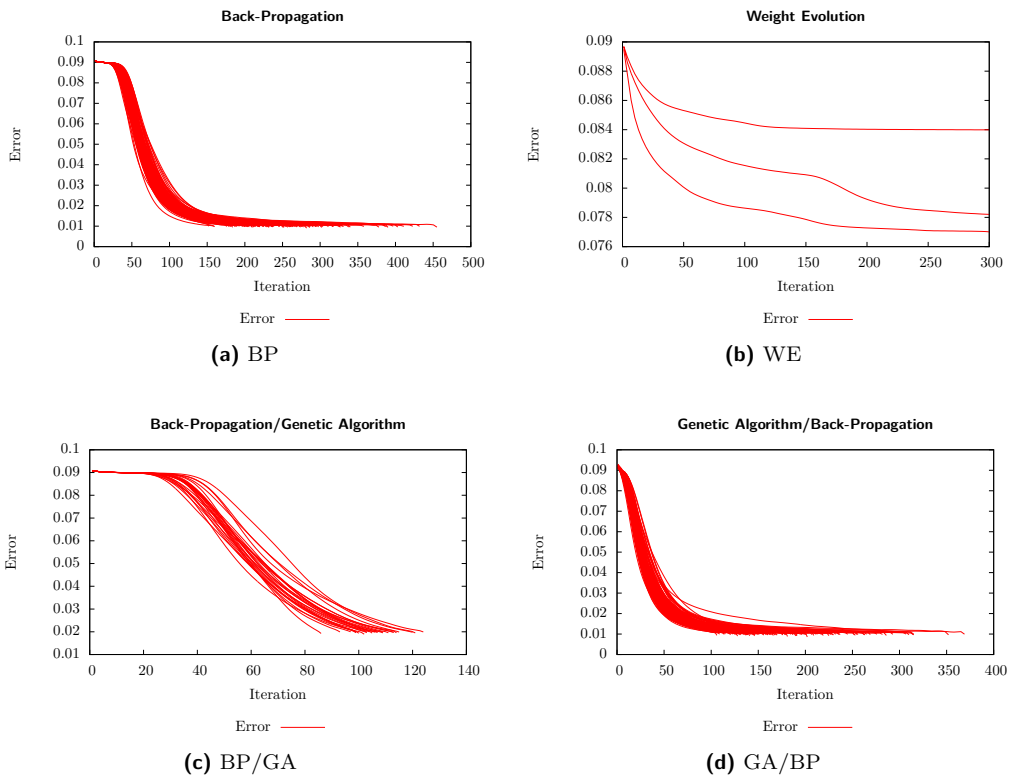


Figure 4.1: The training phases with all available data.

Even though BP/GA is slow, WE is far worse; one run takes roughly 1.5 weeks. WE does not scale well, mainly because of the repeated error calculation and weight evolution. This is, however, not the only reason why WE is slower than the rest. Evolution is generally slow, but the GA phases in BP/GA and GA/BP are threaded. Since the simulations were run on a computer with up to 20 available cores, both BP/GA and GA/BP were run in parallel. As the evolution in WE is smaller and more tightly bound to the rest of the algorithm, no such threading was implemented.

The slowness of WE is not only regarding how long one iteration takes to run, but also regarding the decrease in error during one iteration. While BP and both the BP parts of BP/GA and GA/BP use somewhere between 50 and 140 iterations to achieve an error below 0.02, WE uses 300 iterations and barely achieved an error below 0.077 (Figure 4.1 on the preceding page). By that time, the decrease in error has begun to stagnate.

It is noteworthy, however, that in the two-phase algorithms BP/GA and GA/BP, the latter algorithm is reducing the error faster than it would have without the help of the former. For example, the error in the BP part of GA/BP (Figure 4.1d) decreases faster than in both BP (Figure 4.1a) and the BP part of BP/GA (Figure 4.1c).

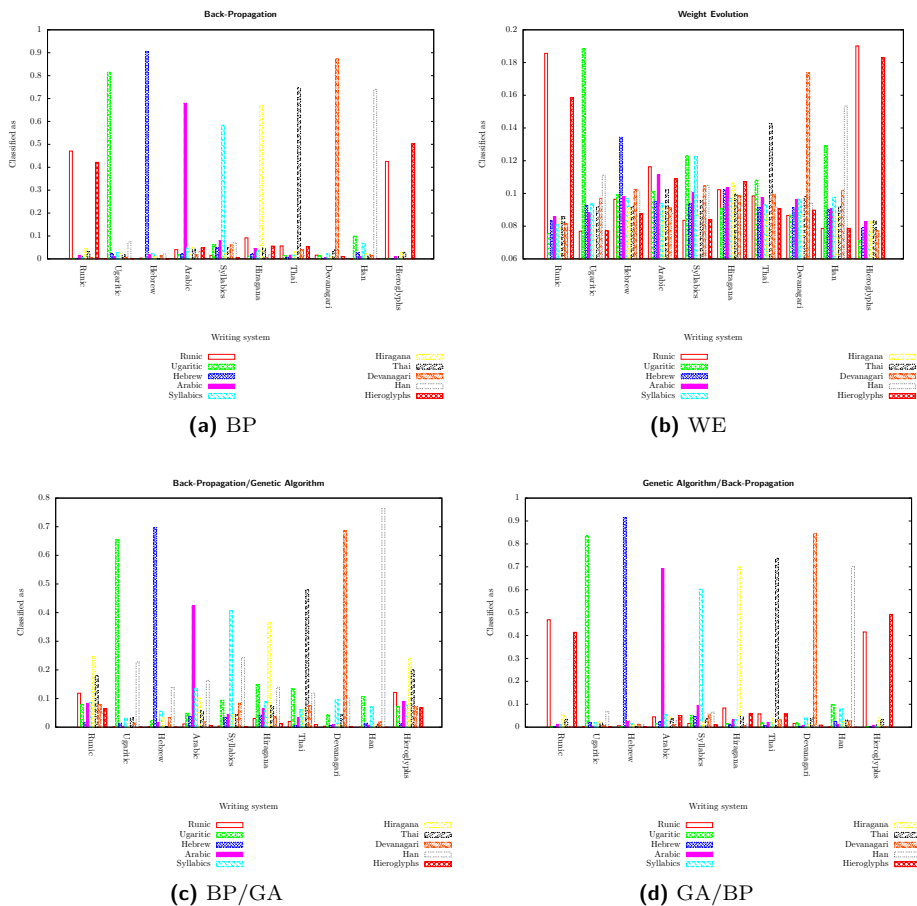


Figure 4.2: The testing phases with all available data.

4.1.2 Testing

With regard to the errors when the learning stops, the test results are of no surprise (Figure 4.2 on the preceding page and Figure 4.3). As WE only got below an error of 0.077, it did not manage to separate between the writing systems in the test phase. In certain cases, e.g. with Runic, Ugaritic, Hebrew, Thai, Devanagari, and Han, the majority of glyphs are falsely classified. However, they are more often classified as the correct writing system than as any specific other writing system.

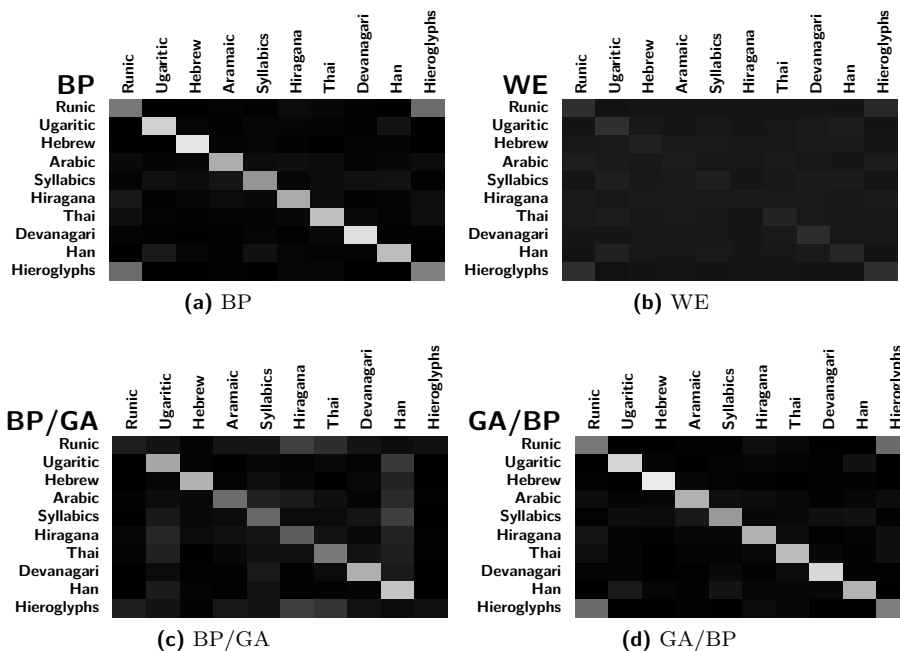


Figure 4.3: The testing phases with all available data.

The results are improved using BP/GA. Ugaritic and Devanagari have also here a high success rate, and Hebrew and Han have improved significantly. Arabic, Canadian Aboriginal Syllabics, Hiragana, and Thai are misclassified more than 50% of the time, but they are more often classified as themselves than as a specific other. The only real problem with BP/GA is the classification of Runic and Hieroglyphs. These two writing systems have similar classification patterns, where both are more often classified as Runic than as Hieroglyphs. To make it even worse, both are more often classified as Devanagari or Thai than as Runic or Hieroglyphs.

GA/BP and BP have similar results, both far better than BP/GA and especially WE. Hebrew has over 90% correct classification, with Ugaritic and Devanagari above 80% correct. Han — which had such good results in BP/GA — has a decreased success rate, pushing it out of the top four. Arabic, Syllabics, Hiragana, and Thai are significantly improved, all being correctly classified more than 50% of

the time. Runic and Hieroglyphs are also here problematic; they are classified as either Runic or Hieroglyphs, with the correct scoring barely better than the other. These two writing systems tend to be problematic, as both their glyphs are drawn with lighter colours than the rest of the writing systems (Rødland, 2010).

The results are plainly visible in Figure 4.3 on the preceding page: both BP and GA/BP have a clear, white diagonal (Figure 4.3a and Figure 4.3d). BP/GA also has a distinct diagonal, but it is greyer than the two other. There is also more general noise (Figure 4.3c). WE has only a slightly visible diagonal, and consists of mostly noise (Figure 4.3b).

4.2 Writing System Comparison

Table 4.1: This table summarizes the success rates of the different writing systems for each of the algorithms, ranging from 0 to 1. The right-hand average is the ratio of correctly classified glyphs for each of the writing systems. This ratio says something about how well the writing system is classified, independent of algorithm. The bottom average is the ratio of correctly classified glyphs for each of the writing systems. This is a measure of how well the algorithms do the classification. The double average in the bottom right corner is the overall ratio of correctly classified glyphs, also this ranging from 0 to 1.

Writing system	BP	WE	BP/GA	GA/BP	Average
Runic	0.470743	0.185628	0.118436	0.468775	0.310895
Ugaritic	0.814603	0.188554	0.653775	0.834615	0.622886
Hebrew	0.904731	0.134322	0.697757	0.915753	0.663140
Arabic	0.680462	0.111346	0.424183	0.692565	0.477139
Syllabics	0.583577	0.122519	0.407675	0.602064	0.428958
Hiragana	0.668136	0.106433	0.364232	0.699212	0.459503
Thai	0.747521	0.142731	0.479718	0.737147	0.526779
Devanagari	0.874799	0.173601	0.685202	0.846093	0.644923
Han	0.740228	0.153396	0.763770	0.701455	0.589712
Hieroglyphs	0.503152	0.182896	0.068356	0.492199	0.311650
Average	0.698795	0.150142	0.466310	0.698987	0.503559

The ratios of correctly classified glyphs can be found in Table 4.1. This table supports the otherwise obvious fact that BP and GA/BP are better at classifying than BP/GA is, and that BP/GA still is far better than WE. More surprisingly, Table 4.1 reveals that GA/BP — with 69.90% correctly classified glyphs — actually is a notch better than BP, with an average of 69.88%. This is not a statistically significant difference, especially since BP is better than GA/BP in 4 of the ten writing systems. There are in general many similarities between BP and GA/BP, probably because the few generations of GA at the beginning of GA/BP do not affect the final result that much. They have roughly the same percentage of correctly

classified glyphs and share many of the same misclassifications.

One misclassification shared by all four algorithms is the confusion around Runic and Hieroglyphs. This is due to the program used to create the glyph images; while the other writing systems are drawn using black on white, Runic and Hieroglyphs are drawn using a grey-scale. Consequently, the glyphs are deemed more similar than they actually are (Rødland, 2010), which also can be seen in Figure 3.1 on page 27. Because of this similarity, Runic and Hieroglyphs are at the bottom of the list with respect to correct classification (Table 4.1 on the preceding page). However, Runic and Hieroglyphs are among the best classified systems using WE. As these two are confused with each other, they stand out from the rest, making it easier to not misclassify them.

There seems to be no similarity between Hiragana and Han (Figure 4.3 on page 50). These two writing systems have a special relationship, as Hiragana is a simplification of Han (Section 3.1.6). It is thus interesting to note that the system has no problem separating the two. This suggests that the system is more interested in the ratio of dark and light nodes — as there are more black pixels in Han glyphs than in Hiragana glyphs (Appendix A) — than in shapes and curves.

4.2.1 Comparing Classification Results with Glyph Appearance

There seems to be a relation between the shape of the glyphs and the classification results. Hebrew, Devanagari, and Ugaritic are the writing systems with the best classification results (Table 4.1 on the preceding page). These are also among the most recognizable systems. Almost all Devanagari glyphs have the same horizontal line (Figure A.8 on page A-9), such that the only input nodes needed to classify Devanagari are the nodes representing those pixels. There is not a 100% correct classification, however, as also other glyphs have a horizontal line in this area. Also Hebrew glyphs follow mainly the same stroke pattern (Figure A.3 on page A-4), and should thus be relatively easy to classify. The same goes for Ugaritic (Figure A.2 on page A-3), as the glyphs consist of the same wedges, going in mostly the same directions.

Han and Thai are not doing as well, but has still more than 50% correct classification in average, WE included (Table 4.1 on the preceding page). The Thai glyphs share many of the same curves and shapes (Figure A.7 on page A-8). Humans can recognize them due to the little circles, but that is probably too much to ask of the artificial neural network (ANN). However, most glyphs have a rounded line from top left and down the right edge. This could be used to recognize the glyphs. The Han glyphs are also recognizable, as many of them have a high stroke count, and thus a high black/white ratio (Figure A.9 on page A-10). The Egyptian Hieroglyphs (Figure A.10 on page A-11) are also rather complex; however, it is easy to separate between Han and Hieroglyphs, as the Hieroglyphs are drawn in a grey-scale and thus have different pixel values.

The worst classified writing systems — ignoring Runic and Hieroglyphs due to their grey-scale problem — are Canadian Aboriginal Syllabics, Hiragana, and Arabic. The Syllabic glyphs consist mainly of the same shapes, only rotated (Figure A.5 on page A-6). One would think these would be easy to classify, but as

the rotation shifts the black and white pixels, the network is unable to recognize the shapes. Syllabics could probably profit from a shape-recognizing preprocessing layer, as the current network seems to be unable to correctly recognize shapes.

One would think Hiragana did well, as many of its glyphs — or *kana* — are more or less identical. That is, many of the kana have diacritics (Figure A.6 on page A-7), converting e.g. は (*ha*) to ば (*ba*) or even ぱ (*pa*). Some of the kana are also available in smaller versions, to replace the vocal sound in other kana. E.g., や (*ya*) is written や when it follows e.g. に (*ni*), creating にや (*nya*) (Banno et al., 1999). As a result, most of the kana have at least one other version, either with diacritics or resized. However, this does not help much. After all, the small versions do not match the pixels of the larger versions, thus making them useless without a well-functioning shape-recognizing network. The kana with diacritics, however, should trigger almost precisely the same pixels. Unfortunately, it seems two similar glyphs of the 30 used is not enough to give great results. In addition, if all versions of the glyph are in the same set (training or test set), the similarity will not help the classification at all. Another point is that — since Hiragana are simplifications of different kanji — the different Hiragana do not look very alike. They are similar in that they have the same stroke appearance, looking like they are drawn by a paintbrush. However, this does not directly affect the pixels, and are thus not discovered by the network.

4.3 Potential Bias Problems

As seen in Figure 4.2 on page 49, BP and GA/BP have achieved significantly better results than BP/GA, which again is doing far better than WE. These results are averaged over all the test runs, of which there are 120 runs of BP and GA/BP, 25 of BP/GA, and 3 of WE. In other words, there are significantly more runs of BP and GA/BP than of BP/GA, and again far more runs of BP/GA than of WE. This could be some of the reason why BP and GA/BP achieve so much better results than the algorithms with longer runtime.

To test the hypothesis, result graphs were generated using (1) the first 25 runs, and (2) the first 3 runs. These can be found in Section B.3.

The point of the first test is to use enough runs to make it statistically significant, but no more. Consequently, only the first 25 runs were used to generate the graphs. However, as there only are 3 available runs of WE, WE would still be in the minority. To solve this, another test was performed using only the first 3 runs. This would not be statistically significant, but at least it would give all the algorithms the same base of insignificance. As a result, the test with 25 runs will be used to investigate whether BP/GA still is significantly worse than BP and BP/GA, while the test with only 3 runs investigates WE.

4.3.1 Test with 25 Runs

As the error functions in all the runs in BP, BP/GA and GA/BP were close (Figure 4.1 on page 48), there is no important difference between these training

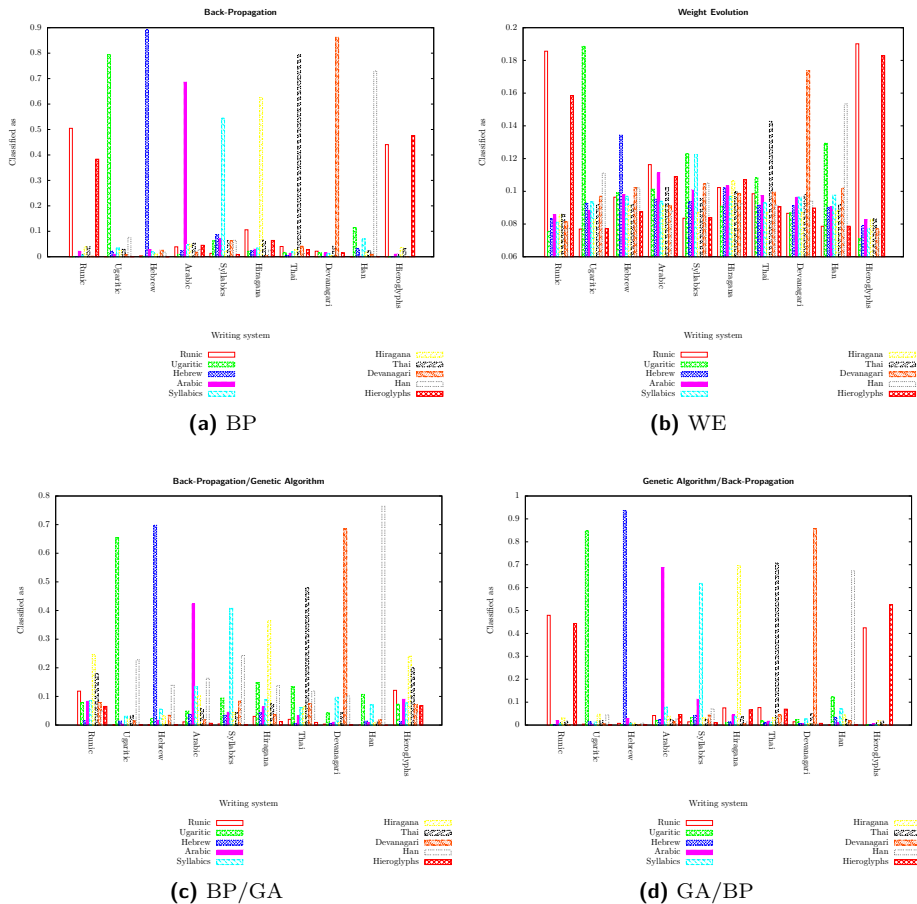


Figure 4.4: The testing phases with maximum 25 runs.

graphs (Figure B.8 on page B-10) and the full graphs (Figure 4.1 on page 48). They all follow the same main path.

The testing graphs (Figure 4.4) are also very similar to the full graphs (Figure 4.2 on page 49). Some minor differences can be seen, e.g. on Runic classification using BP (Figure 4.4a and Figure 4.2a). This difference is far from significant, and it suggests that the number of runs have had little impact on the test results. This is supported by the matrix graphs (Figure B.10 on page B-12 and Figure 4.3 on page 50). Even with only 25 runs, BP and GA/BP are still far better than both BP/GA and WE.

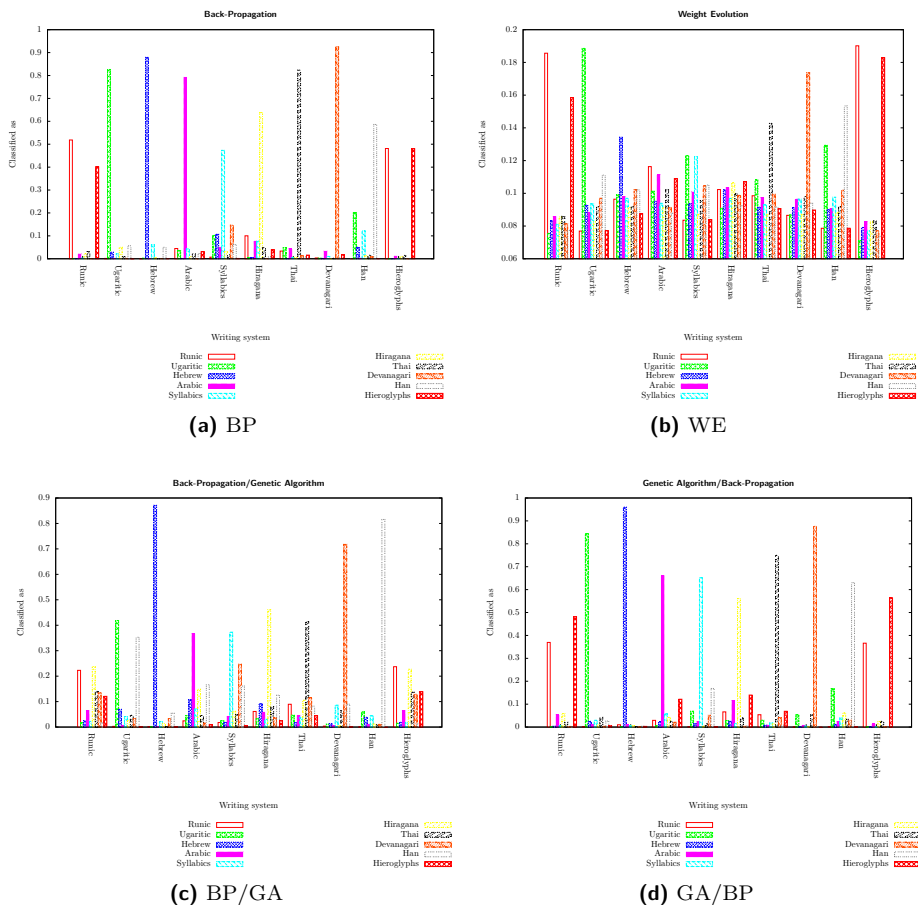


Figure 4.5: The testing phases with maximum 3 runs.

4.3.2 Test with 3 Runs

The error functions in the first 3 runs (Figure B.11 on page B-13) are not significantly different from those in the other runs (Figure 4.1 on page 48). The test graphs, however, are interesting. The difference between the first 3 runs (Figure 4.5) and all the runs (Figure 4.2 on page 49) is clearly visible; however, the test graphs from BP (Figure 4.5a), BP/GA (Figure 4.5c), and GA/BP (Figure 4.5d) using only the first 3 runs are far better than the WE graph (Figure 4.5b). They are not as good as the original; Hieroglyphs are too often misclassified as Runes in BP, Runes are too often misclassified as Hieroglyphs in GA/BP, and the success rates are generally too low in BP/GA. However, in certain cases the results are better than the original, e.g. the classification of Hebrew in GA/BP, the classification of Arabic, Thai, and Devanagari in BP, and Hebrew and Han in BP/GA.

This can also be seen in Figure B.13 on page B-15, where all algorithms but WE display the white diagonal, signifying correct classification. The results of BP, BP/GA and GA/BP are thus clearly better than those of WE, even with only 3 runs.

4.3.3 Conclusion

There seems to be no correlation between the number of runs used as data, and the test results. The hypothesis was that all the small errors from each of the runs were averaged out in BP and GA/BP, but stuck in BP/GA and WE. This does not seem to be the case, as BP and GA/BP achieved good results even when using only 3 runs as basis for the graphs.

If runtime is connected to the classification capabilities of the algorithms, it is because the slower algorithms do not run long enough to achieve an acceptably low error, not because of the smaller number of runs. BP and GA/BP are thus better than BP/GA, and BP/GA is better than WE, independent of how many times the algorithms are run.

CHAPTER 5

Conclusion

This chapter will conclude the dissertation. The research questions are examined, and hopefully answered satisfactorily. In addition, the future of both the glyph classification problem and generalizing artificial neural networks (ANNs) are discussed.

5.1 Concluding the Research Questions

This dissertation tries to answer two research questions (Section 1.2.1):

- RQ1** Are ANNs able to generalize over glyphs of many different writing systems, and correctly classify the writing system of an unseen glyphs?
- RQ2** Which training method would best solve RQ1; learning, evolution, or a hybrid of learning and evolution?

These will be discussed and concluded here; RQ1 in Section 5.1.1 and RQ2 in Section 5.1.2.

5.1.1 RQ1: Generalization

The main question behind this dissertation was if ANNs are able to generalize in such an extent that they can recognize the writing system of an unseen glyph, based only on training on other glyphs from said writing system. It seems that they are in fact capable of this. The results are not perfect, but some of the ANNs are able to connect most of the unseen glyphs to the correct writing system (Figure 4.2 on page 49).

It is unknown *why* this generalization actually works. As discussed in Section 4.2.1, the ANNs do not appear to recognize shapes. There is no preprocessing layer that recognizes the shapes or curves, and it does not appear to be any emerging shape recognition. That is, the recurring shapes in Canadian Aboriginal Syllabics and Hiragana do not seem to be recognized. As the writing systems with the worst classification results are those with the most prominent shapes, it appears that the ANNs do not do their classification based on said shapes.

Instead, the ANNs appear to use the pixel values directly. The best classified writing systems — i.e. Hebrew, Devanagari and Ugaritic — are those with the most recognizable shapes. Most Hebrew glyphs follow the same stroke pattern (Figure A.3 on page A-4), almost all Devanagari glyphs have the same horizontal line (Figure A.8 on page A-9), and the Ugaritic glyphs are all composed of the same wedges (Figure A.2 on page A-3). In other words, they tend to have the same black pixels, and are thus firing the same input nodes.

The good classification of Han, whose glyphs generally have a high stroke count and thus fire many input neurons, suggests that also the amount of fired neurons are important to the generalization.

This suggests that the ANNs are basing their classification directly on the pixel values, without any emerging generalization per se. The important features are not shapes and curves, but the exact pixels that are coloured, and how many of the pixels that are coloured. The approximate colour of the coloured pixels is apparently also important, as the ANNs struggle to separate between the grey-scale Hieroglyphs and Runic (Section 4.2).

The topic of this dissertation was chosen with the hope that ANNs were able to generalize over glyphs and recognize the shapes and curves common for the writing systems (Section 1.1.2). It appears that this is not possible, at least not without a preprocessing layer. However, the ANNs do seem to generalize nevertheless. This might suggest that the pixels alone are enough to classify writing systems.

Is this form of generalization enough to conclude that ANNs can generalize? Even though they do not recognize the shapes of the glyphs, they *are* able to generalize enough over some of the glyphs to recognize the rest of the glyphs. It can thus be concluded that ANNs are able to generalize sufficiently for the glyph classification problem, but that three-layered ANNs are not able to recognize shapes.

5.1.2 RQ2: Method

The results — found in e.g. Figure 4.2 on page 49 and Table 4.1 on page 51 — clearly reveal that back-propagation (BP) and genetic algorithm/back-propagation (GA/BP) are the superior methods. As GA/BP is BP preceded by only 10 iterations of a genetic algorithm (GA), both these methods are mainly BP. One way to look at this is that BP solves the problem, and that GA/BP either does not contain enough GA to ruin the network, or that the BP part of GA/BP manages to salvage the network after the GA is done with it.

There are no particular differences between GA/BP and BP in neither classification results nor runtime. The learning part of GA/BP is faster than that of

BP, but in return the GA part takes up precious time. The only real difference is concerning implementation; BP is a simple back-propagation algorithm, while GA/BP is BP *and* a GA. In other words, GA/BP necessitates roughly twice as much implementation and tuning, but without gaining any advantages. BP seems thus to be the best and simplest solution.

The glyph classification problem is a complex problem, requiring a large network. This large network is probably the reason why evolution is a suboptimal solution for this problem. In most papers advocating evolution of ANNs, the networks evolved are small, with 20 to 50 nodes (Yin et al., 2011; Cao and Jin, 2007; Osman et al., 2010; Su et al., 2009). It is a large difference between a network of maximum 50 nodes and a network with 460 nodes, especially when comparing the number of weights needed. Assuming a full topology with 30 input nodes, 20 hidden nodes and 10 output nodes, there are 800 weights to evolve. The glyph classification network has — with its 401 input nodes, 51 hidden nodes and 10 output nodes — 20961 weights. This is a considerable difference, that will greatly affect the runtime.

When the number of weights in an evolutionary network is increased, one of two things must happen; (1) the size of the genome must increase due to additional genes, or (2) the allocated number of bits per gene must decrease. If the genome size is increased, the total runtime will increase. If one instead decreases the gene size, the precision of the weight value will decrease, making it difficult to fine-tune the weights. In that case, the perfect weight value might be outside of what the gene is able to represent. There exists an ANN that is able to solve the glyph classification problem, as found by BP. However, the low precision of GA weights might make it impossible to find this ANN using a GA. So either way, evolution suffers when the ANN grows. The runtime of BP is also increased when the ANN grows; however, BP scales better than evolution, mainly because it only works with one ANN at a time.

None of these algorithms are as fine-tuned as they ought to be. The parameters in Table 3.2 on page 36 are found by trial and error, but not intensive trial and error. The main problem with parameter tuning is the runtime of the algorithms. For example, weight evolution (WE) has a runtime of 1.5 weeks, making it difficult to test that many parameter value combinations. This is also a problem with back-propagation/genetic algorithm (BP/GA) and GA/BP; even though their runtime is shorter, they have considerably more parameters to tune due to the GA. These parameters are thus far from perfect, but the best tuning would allow given the time frame. There might exist a set of weights that would make the GA solve the problem faster and/or better. However, it would take a lot of time and tweaking to find this set of parameters, and it is not even certain that it exists. This is another huge advantage with BP compared to any evolutionary algorithm; there is only one parameter to tweak, i.e. the learning rate η . As the runtime of the BP is rather fast, it is simple to find the perfect parameter value. This value is the only thing that is needed; having that, one can release the algorithm and it will find the solution without additional frills. As a result, BP is both faster to implement, faster to prepare, and faster to run than any of the other three algorithms. In

addition, it achieves the best results.

When working with such large networks, it is important to keep things simple. While GA is unsuccessful due to the size of the ANN, BP manages to solve the problem within reasonable time. Their hybrids are suboptimal compared to the pure BP. A standard BP is thus the optimal solution; simple, fast, and effective.

Why are none of the hybrids as suitable for this task as the BP? Probably because the hybrids contain evolution. Evolution is already shown to be too time-consuming due to the size of the network (Rødland, 2010), and it suffers from parameters that are difficult and time-consuming to perfect. However, even though the GA does not improve on the BP, the hybrids do solve the memory problems of the pure GA (Rødland, 2010). Consequently, the BP is improving the GA. This is probably because BP reduces the amount of GA needed, such that the system never reaches the critical state.

5.2 Future Work

There is still much work that can be done on the glyph classification problem, both concerning generalization and the actual classification.

5.2.1 Improving the Generalizing ANN

To improve the generalization, one could investigate other algorithms to fine-tune ANNs. At the moment, only BP learning and evolution are examined; there might be other algorithms that will do an even better job than BP did.

Evolutionary algorithms will probably not work, due to the size of the network. However, other algorithms might improve the results, e.g. gradient descent, simulated annealing (SA), or expectation-maximization (EM). One could also try unsupervised or reinforcement learning, but this would probably not work as well as supervised learning, as this is a classification task.

5.2.2 Improving the Glyph Classification

If the point is to make a system that can recognize the writing system of an unseen glyph — without proving anything with respect to generalization capabilities — there are many things that can be tested.

The current network does not seem to have shape recognition capabilities. This could be improved by e.g. adding a preprocessing layer that uses image processing techniques to extract the shapes, lines and curves. If the input to the network would be information about the shapes and lines instead of all the 400 pixels, the number of input nodes would be greatly decreased. This would again reduce the number of weights, and thus reduce the runtime. This network could probably be small enough for evolution to work.

The number of hidden nodes can be optimized by changing the network topology, e.g. by adding or removing hidden nodes (Castillo-Valdivieso et al., 2002;

MacLeod and Maxwell, 2001; Islam et al., 2009; Abraham and Nath, 2000). However, evolution should not be attempted as long as there are 400 input nodes in the network, so this should be tried in addition to something that requires a different kind of input.

The ANN could also be fortified with another hidden layer. This might improve the shape recognition capabilities of the ANN, as the extra layer could be used to represent different shapes. This should not be used in combination with evolution, however, due to the runtime issues.

A simpler improvement might be to continue tweaking the current BP algorithm. There are limitations to the usefulness of this, but some improvements might be achieved.

One could also revisit BP/GA and GA/BP, and find a smoother transition between the BP and GA phases. This will probably not do wonders, but could improve at least BP/GA a bit. That is, if one manages to reduce the data loss in the transition from BP to GA, the GA might get a head start.

One could also try to build one sub-network for each writing system, and merge them afterwards. This would decrease the size of the network, and might thus work with evolution. Most important, it would create sub-networks specified for each writing system, hopefully improving the classification capabilities.

5.2.2.1 The Future of Glyph Classification

There is a huge difference in performance depending on the number of writing systems used in the classification, as shown by Rødland (2010). When taking the huge amount of existing writing systems into account — there are about 1000 writing systems featured in (Ager, 2011) — it is obvious that the glyph classification problem is a lot harder in reality than assumed here. For this to be anything but an excuse to test generalization, one should allow for glyphs from any writing system. This will entail huge training and test sets, which will drastically increase the runtime. It would also necessitate one output node for each writing system, i.e. almost 1000 output nodes. This will lead to a gigantic ANN, which also will drastically increase the runtime. To top it off, it will probably achieve very suboptimal results, based on the experiences from Rødland (2010). It would also suffer from similar-looking writing systems, which further complicates the classification.

An educated guess thus suggests that a fully functional glyph classification system never will see the light of day. For it to work, one would need at least (1) very impressive hardware, to make up for the increase in runtime; (2) another way to match the output nodes to the correct writing system, to reduce the number of output nodes; and (3) a way to preprocess the glyphs, improving the shape recognition, to reduce the number of input nodes and improve the classification.

That is not to say that glyph classification is completely useless. It could be used in e.g. Japan and Korea, where multiple writing systems are in use. In that case, one could send text through writing system classification before sending it through a character recognition program, specialized on the classified writing system. That way, one could use existing software for character recognition to machine translate the text, even though the text consists of multiple writing systems. But if

these possible writing systems are not known in advance, this would be an almost impossible task, necessitating the omniscient glyph classification system described above.

Glyph classification will never be perfect; however, the classification capabilities ought to be acceptable given a sufficiently reduced set of potential writing systems.

References

- Ajith Abraham and Baikunth Nath. Optimal Design of Neural Nets using Hybrid Algorithms. *PRICAI 2000 Topics in Artificial Intelligence*, pages 510–520, 2000. URL <http://www.springerlink.com/index/d426j54323764285.pdf>.
- Simon Ager. Omniglot - writing systems and languages of the world, 2011. URL <http://www.omniglot.com>.
- M. Altan. Reducing Shrinkage in Injection Moldings via the Taguchi, ANOVA and Neural Network Methods. *Materials & Design*, 31:599–604, 2010.
- Krzysztof Bandurski and Wojciech Kwedlo. A Lamarckian Hybrid of Differential Evolution and Conjugate Gradients for Neural Network Training. *Neural Processing Letters*, 32(1):31–44, June 2010. ISSN 1370-4621. doi: 10.1007/s11063-010-9141-1. URL <http://www.springerlink.com/index/10.1007/s11063-010-9141-1>.
- Eri Banno, Yutaka Ohno, Yoko Sakane, and Chikako Shinagawa. *Genki 1: An Integrated Course in Elementary Japanese*. 1999. ISBN 4789009637.
- E.B. Baum, D. Boneh, and C. Garrett. Where Genetic Algorithms Excel. *Evolutionary Computation*, 9(1):93–124, 2001. ISSN 1063-6560. URL <http://www.mitpressjournals.org/doi/abs/10.1162/10636560151075130>.
- Adi Ben-Israel and Thomas N.E. Greville. *Generalized Inverses: Theory and Applications*. Robert E. Krieger Publishing Co., New York City, NY, USA, 1980.
- Christopher M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, Oxford, England, 1996. ISBN 0198538642.
- John A. Bullinaria. John Bullinaria’s Step by Step Guide to Implementing a Neural Network in C, 2009. URL <http://www.cs.bham.ac.uk/~jxb/INC/nn.html>.
- Robert Callan. *The Essence of Neural Networks*. Prentice Hall Europe, Harlow, Essex, England, 1999. ISBN 0-13-908732-X.

- Guangzhen Cao and Ya Qiu Jin. A Hybrid Algorithm of the BP-ANN/GA for Classification of Urban Terrain Surfaces with Fused Data of Landsat ETM and ERS-2 SAR. *International Journal of Remote Sensing*, 28(2):293–305, 2007. URL <http://dx.doi.org/10.1080/01431160500221675>.
- Pedro A. Castillo-Valdivieso, Juan J. Merelo, Alberto Prieto, Ignacio Rojas, and Gustavo Romero. Statistical Analysis of the Parameters of a Neuro-Genetic Algorithm. *Evaluation*, 13(6):1374–1394, 2002.
- C. Charalambous. Conjugate gradient algorithm for efficient training of artificial neural networks. *Circuits, Devices and Systems, IEEE Proceedings G*, 139(3): 301–310, 1992.
- Mark Collier and Bill Manley. *How to read Egyptian hieroglyphs: a step-by-step guide to teach yourself*. University of California Press, Los Angeles, 2003. ISBN 0520239490. URL <http://books.google.com/books?id=aroQ0Zr18J4C&pgis=1>.
- Charles Darwin. *On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life*. Murray, London, England, 1859. URL <http://www.literature.org/authors/darwin-charles/the-origin-of-species/>.
- Kenneth A. De Jong. *Evolutionary Computation : A Unified Approach*. The MIT Press, Cambridge, MA, USA, 2006. ISBN 0-262-04194-4.
- Stanislas Dehaene. *Reading in the Brain : The Science and Evolution of a Human Invention*. Penguin Group, London, England, 2009. ISBN 978-0-670-02110-9.
- W.J. Deng, C.T. Chen, C.H. Sun, W.C. Chen, and C.P. Chen. An Effective Approach for Process Parameter Optimization in Injection Molding of Plastic Housing Components. *Polymer-Plastics Technology and Engineering*, 47:910–919, 2008.
- Y.M. Deng, Y. Zhang, and Y.C. Lam. A Hybrid of Mode-Pursuing Sampling Method and Genetic Algorithm for Minimization of Injection Molding Warpage. *Materials & Design*, 31:2118–2123, 2010.
- Rinku Dewri. Evolutionary Neural Networks: Design Methodologies - AI Depot, 2003. URL <http://ai-depot.com/articles/evolutionary-neural-networks-design-methodologies/>.
- Keith L. Downing. Introduction to evolutionary algorithms. 2009. URL <http://www.idi.ntnu.no/emner/it3708/lectures/notes/evolalgs.pdf>.
- Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification*. 2 edition, 2000.

- R.A. Dunne and N.A. Campbell. On the Pairing of the Softmax Activation and Cross-Entropy Penalty Functions and the Derivation of the Softmax Activation Function. In *8th Australian Conference on Neural Networks*, pages 181–185, Melbourne, Australia, 1997.
- R. Fletcher and C.M. Reeves. Fletcher R, Reeves CM (1964) Function minimization by conjugate gradients. *Comput J* 7:149 – 154. *The Computer Journal*, 7: 149–154, 1964.
- Dario Floreano and Claudio Mattiussi. *Bio-Inspired Artificial Intelligence : Theories, Methods, and Technologies*. The MIT Press, Cambridge, MA, USA, 2008. ISBN 978-0-262-06271-8.
- FOLDOC. Glyph, 1998a. URL <http://foldoc.org/glyphs>.
- FOLDOC. Writing System, 1998b. URL <http://foldoc.org/writing+system>.
- Y.H. Gao and X.C. Wang. An Effective Warpage Optimization Method in Injection Molding Based on the Kriging Model. *International Journal of Advanced Manufacturing Technology*, 37:953–960, 2008.
- Y.H. Gao and X.C. Wang. Surrogate-Based Process Optimization for Reducing Warpage in Injection Molding. *Journal of Materials Processing Technology*, 209: 1302–1309, 2009.
- Yuansheng Huang, Yufang Lin, and Zilong Qiu. Freight Prediction Model Based on GABP Neural Network. In *2008 International Symposium on Computational Intelligence and Design*, pages 229–232, 2008. URL <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4725597>.
- J. Ilonen, J.K. Kamarainen, and J. Lamminen. Differential evolution training algorithm for feed-forward neural networks. *Neural Processing Letters*, 17:93–105, 2003.
- Monirul Islam, Abdus Sattar, Faijul Amin, Xin Yao, and Kazuyuki Murase. A New Constructive Algorithm for Architectural and Functional Adaptation of Artificial Neural Networks. *IEEE Transactions on Systems, Man, and Cybernetics*, 39(6): 1590–1605, 2009.
- M. Tim Jones. *Artificial Intelligence : A Systems Approach*. Jones and Bartlett Publishers, Sudbury, MA, USA, 2009. ISBN 978-0-7637-7337-3.
- H. Kurtaran, B. Ozcelik, and T. Erzurumlu. Warpage Optimization of a Bus Ceiling Lamp Base using Neural Network Model and Genetic Algorithm. *Journal of Materials Processing Technology*, 169:314–319, 2005.
- H. Kurtaran, B. Ozcelik, and T. Erzurumlu. Efficient Warpage Optimization of Thin Shell Plastic Parts using Response Surface Methodology and Genetic Algorithm. *Journal of Materials Processing Technology*, 27:468–472, 2006.

- Jack Kyte and Russell F. Doolittle. A Simple Method for Displaying the Hydrophobic Character of a Protein. *Journal of Molecular Biology*, 157(1):105–132, 1982.
- Chun Lu and Bingxue Shi. Hybrid Back-Propagation/Genetic Algorithm for Multi-layer Feedforward Neural Networks. In *Proceedings of ICSP2000*, pages 571–574, 2000.
- C. MacLeod and G.M. Maxwell. Incremental evolution in ANNs: Neural nets which grow. *Artificial Intelligence Review*, pages 1–32, 2001. URL <http://www.springerlink.com/index/R7T5683401W7368J.pdf>.
- Paul McQuesten and Risto Miikkulainen. Culling and teaching in neuro-evolution. In *Proceedings of the 7th International Conference on Genetic Algorithms*, pages 1–8, 1997. URL <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.49.8014&rep=rep1&type=pdf>.
- Merriam-Webster. Cull, 2010. URL <http://www.merriam-webster.com/dictionary/culling>.
- Sin-Chun Ng and Shu-Hung Leung. A Weight Evolution Algorithm for finding the Global Minimum of Error Function in Neural Networks. In *Proceedings of the 2000 Congress on Evolutionary Computation*, pages 153–157. IEEE, 2000. ISBN 0-7803-6375-2. doi: 10.1109/CEC.2000.870289. URL <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=870289>.
- Eric Francis Osborn. *Clement of Alexandria*. Cambridge University Press, 2005.
- Mohd Haniff Osman, Choong-Yeun Liong, and Ishak Hashim. Hybrid Learning Algorithm in Neural Network System for Enzyme Classification. *International Journal of Advances in Soft Computation and Its Applications*, 2(2): 209–220, 2010. URL <http://www.i-csrs.org/Volumes/ijasca/vol.2/vol.2.2.4.July.10.pdf>.
- Tiril Anette Langfeldt Rødland. *Classifying Glyphs: Comparing Evolution and Learning*. Master project, NTNU, 2010. URL http://www.pvv.ntnu.no/~tirilane/glycla_project.pdf.
- D.E. Rumelhart, G.E. Hinton, and R.J. Williams. Learning Representations by Back-Propagation of Errors. *Nature*, 1986.
- C.Y. Shen, L.X. Wang, and Q. Li. Optimization of Injection Molding Process Parameters using Combination of Artificial Neural Network and Genetic Algorithm Method. *Journal of Materials Processing Technology*, 183:412–418, 2007.
- Min Shi and Haifeng Wu. Evolving Efficient Connection for the Design of Artificial Neural Networks. In *Proceedings of the 18th International Conference on Artificial Neural Networks*, pages 909–918, Trondheim, Norway, 2008. URL <http://www.springerlink.com/content/au45qn6753k42360/fulltext.pdf>.

- R. Storn and K. Price. Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11: 341–359, 1997.
- Jianmin Su, Bifeng Song, and Baofeng Li. *A Hybrid Algorithm of GA Wavelet-BP Neural Networks to Predict Near Space Solar Radiation*, chapter A Hybrid A, pages 442–450. Springer, Berlin, 2009. URL <http://www.springerlink.com/content/d2k18mr24118v248/fulltext.pdf>.
- The Unicode Consortium. *The Unicode Standard, Version 5.2.0*. The Unicode Consortium, Mountain View, CA, USA, 2009. ISBN 978-1-936213-00-9. URL <http://www.unicode.org/versions/Unicode5.2.0/>.
- The Unicode Consortium. *The Unicode Standard, Version 6.0.0*. The Unicode Consortium, Mountain View, CA, USA, 2011. ISBN 978-1-936213-01-6. URL <http://www.unicode.org/versions/Unicode6.0.0/>.
- Stephen D. Turner, Scott M. Dudek, and Marylyn D. Ritchie. ATHENA: A knowledge-based hybrid backpropagation-grammatical evolution neural network algorithm for discovering epistasis among quantitative trait Loci. *Bio-Data mining*, 3(1):5, January 2010. ISSN 1756-0381. doi: 10.1186/1756-0381-3-5. URL <http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=2955681&tool=pmcentrez&rendertype=abstract>.
- Kiri Wagstaff. ANN Backpropagation : Weight updates for hidden nodes. pages 1–3, 2008. URL <http://www.wkiri.com/cs461-w08/Lectures/Lec4/ANN-deriv.pdf>.
- D. Whitley, T. Starkweather, and C. Bogart. Genetic algorithms and neural networks: optimizing connections and connectivity. *Parallel Computing*, 14(3): 347–361, August 1990. ISSN 01678191. doi: 10.1016/0167-8191(90)90086-O. URL <http://linkinghub.elsevier.com/retrieve/pii/0167819190900860>.
- B. Widrow and M.E. Hoff. Adaptive Switching Circuits. In *Proceedings of the 1960 IRE WESCON Convention*, pages 96–104, New York City, NY, USA, 1960. IRE.
- Wikibooks. Artificial Neural Networks/Activation Functions - Wikibooks, collection of open-content textbooks, 2010. URL http://en.wikibooks.org/w/index.php?title=Artificial_Neural_Networks/Activation_Functions&oldid=1942655.
- Fei Yin, Huajie Mao, and Lin Hua. A Hybrid of Back Propagation Neural Network and Genetic Algorithm for Optimization of Injection Molding Process Parameters. *Materials & Design*, pages 1–8, February 2011. ISSN 02613069. doi: 10.1016/j.matdes.2011.01.058. URL <http://linkinghub.elsevier.com/retrieve/pii/S0261306911000781>.
- Y. Zhang, Y.M. Deng, and B.S. Sun. Injection Molding Warpage Based on a Mode-Pursuing Sampling Method. *Polymer-Plastics Technology and Engineering*, 48: 767–774, 2009.

Software Library

alexandria. Alexandria. <http://www.cliki.net/Alexandria>.
[http://common-lisp.net/~loliveira/tarballs/inofficial/
alexandria-2008-07-29.tar.gz](http://common-lisp.net/~loliveira/tarballs/inofficial/alexandria-2008-07-29.tar.gz), version 2008-07-29.

atdoc. atdoc : Common Lisp documentation generation. [http://
www.lichteblau.com/atdoc](http://www.lichteblau.com/atdoc). [http://www.lichteblau.com/atdoc/download/
atdoc.tar.gz](http://www.lichteblau.com/atdoc/download/atdoc.tar.gz), version 2008.11.30. David Lichteblau.

babel. Babel. <http://www.cliki.net/Babel>. [http://common-lisp.net/
project/babel/releases/babel_latest.tar.gz](http://common-lisp.net/project/babel/releases/babel_latest.tar.gz), version 0.3.0.

ch-image. ch-image. <http://www.cliki.net/ch-image>. [http://cyrusharmon.
org/static/releases/ch-image_0.4.1.tar.gz](http://cyrusharmon.org/static/releases/ch-image_0.4.1.tar.gz).

ch-util. ch-util. [http://cyrusharmon.org/static/releases/ch-util_0.3.10.
tar.gz](http://cyrusharmon.org/static/releases/ch-util_0.3.10.tar.gz).

cl-jpeg. cl-jpeg. <http://www.cliki.net/cl-jpeg>. [http://www.common-lisp.
net/project/mcclim/cl-jpeg.tar.gz](http://www.common-lisp.net/project/mcclim/cl-jpeg.tar.gz).

cl-ppcre. CL-PPCRE : Portable perl-compatible regular expressions for Common Lisp. <http://www.weitz.de/cl-ppcre>. Ubuntu package `cl-ppcre`, [http://
weitz.de/files/cl-ppcre.tar.gz](http://weitz.de/files/cl-ppcre.tar.gz), version 2.0.1-2. Edi Weitz.

cl-unicode. CL-UNICODE : A portable unicode library for Common Lisp. [http://
www.weitz.de/cl-unicode](http://www.weitz.de/cl-unicode). <http://weitz.de/files/cl-ppcre.tar.gz>, ver-
sion 0.1.1. Edi Weitz.

cl-yacc. CL-Yacc. <http://www.cliki.net/CL-Yacc>. [http://www.pps.jussieu.
fr/~jch/software/files/cl-yacc-0.3.tar.gz](http://www.pps.jussieu.fr/~jch/software/files/cl-yacc-0.3.tar.gz), version 0.3.

clem. clem. http://cyrusharmon.org/static/releases/clem_0.4.1.tar.gz.

closer-mop. Closer to mop. <http://www.cliki.net/closer-mop>. [http://www.
common-lisp.net/project/closer/ftp/closer-mop_latest.tar.gz](http://www.common-lisp.net/project/closer/ftp/closer-mop_latest.tar.gz), version
0.61.

closure-common. closure-common. <http://www.cliki.net/closure-common>.
<http://www.common-lisp.net/project/cxml/download/closure-common.tar.gz>, version 2008-11-30.

closure-html. closure-html. <http://www.cliki.net/closure-html>. <http://www.common-lisp.net/project/closure/download/closure-html.tar.gz>,
version 2008-11-30.

cxml. CXML. <http://www.cliki.net/CXML>. <http://www.common-lisp.net/project/cxml/download/cxml.tar.gz>, version 2008-11-30.

cxml-stp. cxml-stp. <http://www.cliki.net/cxml-stp>. <http://www.lichteblau.com/cxml-stp/download/cxml-stp.tar.gz>, version 2008-11-30.

flexi-streams. flexi-streams. <http://www.cliki.net/Flexi-streams>. <http://weitz.de/files/flexi-streams.tar.gz>, version 1.0.7.

pango. Pango. <http://www.pango.org>. Ubuntu packages gir1.0-pango1.0, libpango1.0-0, libpango1.0-0-dbg, libpango1.0-common, libpango1.0-dev, libpango1.0-doc, libsdl-pango-dev, libsdl-pango1, version 1.28.0.

parse-number. Parse-number. <http://www.cliki.net/PARSE-NUMBER>. <http://www.common-lisp.net/project/asdf-packaging/parse-number-latest.tar.gz>, version 1.0.

plexippus-xpath. plexippus-xpath. <http://www.cliki.net/plexippus-xpath>.
<http://www.common-lisp.net/project/plexippus-xpath/download/plexippus-xpath.tar.gz>, version 2008-12-07.

puri. Puri (Portable Universal Resource Identifier). <http://www.cliki.net/Puri>.
<http://files.b9.com/puri/puri-latest.tar.gz>, version 1.5.5.

Salza2. Salza2. <http://www.cliki.net/Salza2>.

sbcl. SBCL (Steel Bank Common Lisp). <http://www.sbcl.org>. Ubuntu package sbcl, version 1.0.29.11.debian.

split-sequence. Split-sequence. <http://www.cliki.net/SPLIT-SEQUENCE>. <http://ftp.linux.org.uk/pub/lisp/experimental/cclan/split-sequence.tar.gz>.

trivial-features. trivial-features. <http://www.cliki.net/trivial-features>.
http://common-lisp.net/~loliveira/tarballs/trivial-features/trivial-features_latest.tar.gz, version 0.5.

trivial-gray-streams. trivial-gray-streams. <http://www.cliki.net/trivial-gray-streams>.
<http://common-lisp.net/project/cl-plus-ssl/download/trivial-gray-streams.tar.gz>, version 2008-11-02.

xuriella. Xuriella. <http://www.clike.net/xuriella>. <http://www.common-lisp.net/project/xuriella/download/xuriella.tar.gz>, version 2009-04-04.

ZPNG. Zpng. <http://www.clike.net/ZPNG>. <http://www.xach.com/lisp/zpng.tgz>.

APPENDIX A

Unicode Charts

Below are examples of glyphs¹ from the ten writing systems used. All the glyphs are not present, as there are over 600 pages in total. This should, however, be enough to get the characteristics of each writing system.

¹Copyright ©1991-2011 Unicode, Inc. All rights reserved. Reproduced with permission of Unicode, Inc.

	16A	16B	16C	16D	16E	16F
0	𐐀 16A0	𐐁 16B0	𐐂 16C0	𐐃 16D0	𐐄 16E0	𐐅 16F0
1	𐐆 16A1	𐐇 16B1	𐐈 16C1	𐐉 16D1	𐐊 16E1	
2	𐐋 16A2	𐐌 16B2	𐐍 16C2	𐐎 16D2	𐐏 16E2	
3	𐐐 16A3	𐐑 16B3	𐐒 16C3	𐐓 16D3	𐐔 16E3	
4	𐐕 16A4	𐐖 16B4	𐐗 16C4	𐐘 16D4	𐐙 16E4	
5	𐐚 16A5	𐐛 16B5	𐐜 16C5	𐐝 16D5	𐐞 16E5	
6	𐐟 16A6	𐐠 16B6	𐐡 16C6	𐐢 16D6	𐐣 16E6	
7	𐐤 16A7	𐐥 16B7	𐐦 16C7	𐐧 16D7	𐐨 16E7	
8	𐐩 16A8	𐐪 16B8	𐐫 16C8	𐐬 16D8	𐐭 16E8	
9	𐐮 16A9	𐐯 16B9	𐐰 16C9	𐐱 16D9	𐐲 16E9	
A	𐐳 16AA	𐐴 16BA	𐐵 16CA	𐐶 16DA	𐐷 16EA	
B	𐐸 16AB	𐐹 16BB	𐐺 16CB	𐐻 16DB	𐐼 16EB	
C	𐐽 16AC	𐐾 16BC	𐐿 16CC	𐑀 16DC	𐑁 16EC	
D	𐑂 16AD	𐑃 16BD	𐑄 16CD	𐑅 16DD	𐑆 16ED	
E	𐑇 16AE	𐑈 16BE	𐑉 16CE	𐑊 16DE	𐑋 16EE	
F	𐑌 16AF	𐑍 16BF	𐑎 16CF	𐑏 16DF	𐑐 16EF	

Figure A.1: Runic. Copyright ©1991-2011 Unicode, Inc. All rights reserved. Reproduced with permission of Unicode, Inc.

































	1038	1039
0	 10380	 10389
1	 10381	 10391
2	 10382	 10392
3	 10383	 10393
4	 10384	 10394
5	 10385	 10395
6	 10386	 10396
7	 10387	 10397
8	 10388	 10398
9	 10389	 10399
A	 1038A	 1039A
B	 1038B	 1039B
C	 1038C	 1039C
D	 1038D	 1039D
E	 1038E	 1039E
F	 1038F	 1039F

Figure A.2: Ugaritic. Copyright ©1991-2011 Unicode, Inc. All rights reserved. Reproduced with permission of Unicode, Inc.

	059	05A	05B	05C	05D	05E	05F
0							
1							
2							
3							
4							
5							
6							
7							
8							
9							
A							
B							
C							
D							
E							
F							

Figure A.3: Hebrew. Copyright ©1991-2011 Unicode, Inc. All rights reserved. Reproduced with permission of Unicode, Inc.

	060	061	062	063	064	065	066	067	068	069	06A	06B	06C	06D	06E	06F
0				ذ	-				پ	ڈ	غ	گ	ة	ي		
1			ء	ر	ف		ا	أ	خ	ز	ف	گ	ه	ي		ا
2			آ	ز	ق		٢	أ	خ	ز	ب	گ	ه	ي		٢
3			أ	س	ك		٣	ا	ج	ر	ب	گ	ه	ي		٣
4			و	ش	ل		٤	ا	ج	ر	ب	گ	ه	ي		٤
5			ا	ص	م		٥	ا	خ	ر	ب	گ	ه	ي		٥
6			ي	ض	ن		٦	و	چ	ر	ب	گ	ه	ي		٦
7			ا	ط	ه		٧	و	چ	ر	ب	گ	ه	ي		٧
8			ب	ظ	و		٨	ي	ذ	ر	ب	گ	ه	ي		٨
9			ة	ع	ي		٩	ث	د	ر	ب	گ	ه	ي		٩
A			ت	غ	ي		ث	د	ب	ن	ك	ن	ق			ش
B			ث	ك			ر	ب	ب	پ	ك	ن	ق			ض
C			ج	ك			ر	ب	ب	پ	ك	ن	ق			غ
D			ح	ي			*	ت	د	ب	پ	ك	ن			ه
E			خ	ي			س	ب	ب	پ	ك	ن				م
F			د	ي			و	ت	ذ	ظ	ك	ب	ف			ه

Figure A.4: Arabic. Copyright ©1991-2011 Unicode, Inc. All rights reserved. Reproduced with permission of Unicode, Inc.

	140	141	142	143	144	145	146	147	148	149	14A	14B	14C	14D
0	◌̇	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́
1	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́
2	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́
3	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́
4	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́
5	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́
6	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́
7	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́
8	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́
9	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́
A	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́
B	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́
C	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́
D	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́
E	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́
F	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́	◌̈́

Figure A.5: Canadian Aboriginal Syllabics. Copyright ©1991-2011 Unicode, Inc. All rights reserved. Reproduced with permission of Unicode, Inc.

	304	305	306	307	308	309
0		ぐ 3050	だ 3060	ば 3070	む 3080	み 3090
1	あ 3041	け 3051	ち 3061	ば 3071	め 3081	ゑ 3091
2	あ 3042	げ 3052	ぢ 3062	ひ 3072	も 3082	を 3092
3	い 3043	こ 3053	っ 3063	び 3073	や 3083	ん 3093
4	い 3044	ご 3054	っ 3064	び 3074	や 3084	う 3094
5	う 3045	さ 3055	づ 3065	ふ 3075	ゆ 3085	か 3095
6	う 3046	ざ 3056	て 3066	ぶ 3076	ゆ 3086	け 3096
7	え 3047	し 3057	で 3067	ふ 3077	よ 3087	
8	え 3048	じ 3058	と 3068	へ 3078	よ 3088	
9	お 3049	す 3059	ど 3069	べ 3079	ら 3089	ゝ 3099
A	お 304A	ず 305A	な 306A	ぺ 307A	り 308A	ゞ 309A
B	か 304B	せ 305B	に 306B	ほ 307B	る 308B	ゝ 309B
C	が 304C	ぜ 305C	ぬ 306C	ぼ 307C	れ 308C	ゞ 309C
D	き 304D	そ 305D	ね 306D	ぽ 307D	ろ 308D	ゝ 309D
E	ぎ 304E	ぞ 305E	の 306E	ま 307E	わ 308E	ゞ 309E
F	く 304F	た 305F	は 306F	み 307F	わ 308F	ゝ 309F

Figure A.6: Hiragana. Copyright ©1991-2011 Unicode, Inc. All rights reserved. Reproduced with permission of Unicode, Inc.

	0E0	0E1	0E2	0E3	0E4	0E5	0E6	0E7
0		๐ 0E10	๑ 0E20	๒ 0E30	๓ 0E40	๔ 0E50		
1	๕ 0E01	๖ 0E11	๗ 0E21	๘ 0E31	๙ 0E41	๐ 0E51		
2	๑ 0E02	๒ 0E12	๓ 0E22	๔ 0E32	๕ 0E42	๖ 0E52		
3	๗ 0E03	๘ 0E13	๙ 0E23	๐ 0E33	๑ 0E43	๒ 0E53		
4	๓ 0E04	๔ 0E14	๕ 0E24	๖ 0E34	๗ 0E44	๘ 0E54		
5	๙ 0E05	๐ 0E15	๑ 0E25	๒ 0E35	๓ 0E45	๔ 0E55		
6	๐ 0E06	๑ 0E16	๒ 0E26	๓ 0E36	๔ 0E46	๕ 0E56		
7	๒ 0E07	๓ 0E17	๔ 0E27	๕ 0E37	๖ 0E47	๗ 0E57		
8	๔ 0E08	๕ 0E18	๖ 0E28	๗ 0E38	๘ 0E48	๙ 0E58		
9	๖ 0E09	๗ 0E19	๘ 0E29	๙ 0E39	๐ 0E49	๑ 0E59		
A	๘ 0E0A	๙ 0E1A	๐ 0E2A	๑ 0E3A	๒ 0E4A	๓ 0E5A		
B	๐ 0E0B	๑ 0E1B	๒ 0E2B		๓ 0E4B	๔ 0E5B		
C	๒ 0E0C	๓ 0E1C	๔ 0E2C		๕ 0E4C			
D	๔ 0E0D	๕ 0E1D	๖ 0E2D		๗ 0E4D			
E	๖ 0E0E	๗ 0E1E	๘ 0E2E		๙ 0E4E			
F	๘ 0E0F	๙ 0E1F	๐ 0E2F	๑ 0E3F	๒ 0E4F			

Figure A.7: Thai. Copyright ©1991-2011 Unicode, Inc. All rights reserved. Reproduced with permission of Unicode, Inc.

	090	091	092	093	094	095	096	097
0	ं 090	ऐ 091	ठ 092	र 093	ी 094	ॐ 095	ऋ 096	० 097
1	ँ 098	ऑ 099	ड 100	ॠ 101	ॡ 102	ॢ 103	ॣ 104	। 105
2	ं 106	ओ 107	ढ 108	ल 109	ॠ 110	ॡ 111	ॢ 112	अँ 113
3	ः 114	ओ 115	ण 116	ळ 117	ॠ 118	ॡ 119	ॢ 120	अ 121
4	ऐ 122	औ 123	त 124	ळ 125	ॠ 126	ॡ 127	। 128	आ 129
5	अ 130	क 131	थ 132	व 133	ँ 134	ँ 135	॥ 136	औ 137
6	आ 138	ख 139	द 140	श 141	ै 142	ॠ 143	० 144	अ 145
7	इ 146	ग 147	ध 148	ष 149	े 150	ॠ 151	१ 152	अ 153
8	ई 154	घ 155	न 156	स 157	ै 158	क 159	२ 160	
9	उ 161	ङ 162	न 163	ह 164	ाँ 165	ख 166	३ 167	ज़ 168
A	ऊ 169	च 170	प 171	ं 172	ो 173	ग 174	४ 175	य 176
B	ऋ 177	ॠ 178	फ 179	ाँ 180	ो 181	ज 182	५ 183	ग 184
C	ॠ 185	ज 186	ब 187	ॠ 188	ौ 189	ड़ 190	६ 191	ज़ 192
D	ँ 193	झ 194	भ 195	ऽ 196	ॠ 197	ढ़ 198	७ 199	१ 200
E	ऐ 201	ज 202	म 203	ाँ 204	ि 205	फ 206	८ 207	ड 208
F	ए 209	ट 210	य 211	ि 212	ौ 213	य 214	९ 215	ब 216

Figure A.8: Devanagari. Copyright ©1991-2011 Unicode, Inc. All rights reserved. Reproduced with permission of Unicode, Inc.

HEX	C	J	K	V	HEX	C	J	K	V
4EA7 ~ B.4	产 G0-327A				4EB9 ~ B.19	亶 亶 亶 亶			
4EA8 ~ B.5	亨 亨 亨 亨 亨 G0-3A60 T1-486C J0-357C K0-7A7B KP0-F3E0 V1-4A5E				4EBA 人 人 人 人 人 G0-484B T1-4429 J0-3F4D K0-6C51 KP0-FCC5 V1-4A62				
4EA9 ~ B.5	亩 G0-4436				4EBB 亻 亻 G0-5859 J4-2135				
4EAA ~ B.5	奕 G1-707D				4EBC 亼 亼 亼 G0-2148 T5-2132 J4-2137				
4EAB ~ B.6	享 享 享 享 享 G0-4F9D T1-4B6A J0-357D K0-7A3D KP0-F3A2 V1-4A5F				4EBD 亼 亼 亼 G1-7D37 K2-2153				
4EAC ~ B.6	京 京 京 京 京 G0-3E29 T1-4B6B J0-357E K0-4C48 KP0-CFAC V1-4A60				4EBE 亼 亼 G0-2149 T5-2133				
4EAD ~ B.7	亨 亨 亨 亨 亨 G0-4D24 T1-4F98 J0-4A02 K0-6F4D KP0-E8BB V1-4A61				4EBF 亿 亿 G0-5D5A H-8902				
4EAE ~ B.7	亮 亮 亮 亮 G0-4141 T1-4F99 J0-4E3C K0-5555 KP0-07DC				4EC0 仵 仵 仵 仵 仵 G0-4432 T1-446F J0-303A K0-6427 KP0-E5EB V1-4A63				
4EAF ~ B.7	盲 盲 盲 盲 G0-2143 T5-2B26 J1-3041 KP1-3492				4EC1 仁 仁 仁 仁 仁 G0-484A T1-446E J0-3F4E K0-6C52 KP0-FCC6 V1-4A64				
4EB0 ~ B.7	京 京 京 京 G0-2144 T5-2B25 J0-5037 K2-2152 KP1-348E				4EC2 仉 仉 仉 仉 G0-586C T2-2132 J0-503E K2-2154 KP1-349E				
4EB1 ~ B.7	徂 徂 徂 徂 G0-2145 T5-2B27 KP1-3493				4EC3 仸 仸 仸 仸 仸 G0-586A T1-4470 J1-3043 K2-2155 KP1-348A V1-4A65				
4EB2 ~ B.7	亲 亲 亲 G0-4757 T5-2B24 KP1-3491				4EC4 仄 仄 仄 仄 仄 G0-5846 T1-4475 J0-503C K0-7631 KP0-EEE1 V1-4A66				
4EB3 ~ B.8	毫 毫 毫 毫 G0-5571 T1-537E J0-5038 K1-6131 KP1-3495				4EC5 仅 仅 G0-3D76 T5-2149 KP1-3440				
4EB4 ~ B.10	亮 亮 G0-2146 T4-3E2E K1-652F				4EC6 仆 仆 仆 仆 仆 G0-494D T1-4471 J0-503D K1-6232 KP1-349F V1-4A67				
4EB5 ~ B.10	褰 G0-5574				4EC7 仇 仇 仇 仇 仇 G0-3370 T1-4472 J0-3558 K0-4E7B KP0-D1E2 V1-4A68				
4EB6 ~ B.11	亶 亶 亶 亶 G0-3257 T2-4158 J0-5039 K0-5322 KP0-DSAE				4EC8 伙 伙 伙 G0-305C T2-2134 J4-2139 KP1-3447				
4EB7 ~ B.11	廉 廉 G0-2147 T3-4033								
4EB8 ~ B.14	廉 弹 H-FBF6 G0-2C22								

Figure A.9: Han. Copyright ©1991-2011 Unicode, Inc. All rights reserved. Reproduced with permission of Unicode, Inc.

	130E	130F	1310	1311	1312	1313	1314	1315	1316	1317	1318	1319	131A	131B
0														
1														
2														
3														
4														
5														
6														
7														
8														
9														
A														
B														
C														
D														
E														
F														

Figure A.10: Egyptian Hieroglyphs. Copyright ©1991-2011 Unicode, Inc. All rights reserved. Reproduced with permission of Unicode, Inc.

APPENDIX B

Result Graphs

This appendix contains all the result graphs, as well as a description on how to read the graphs.

B.1 How to Read the Graphs

This section gives a quick description on how to read the different types of graphs.

B.1.1 Training Graphs

In the training graphs (Figure B.5 on page B-7, Figure B.8 on page B-10, and Figure B.11 on page B-13), the different runs are drawn on top of each other. This helps to point out the similarity of each run within the same algorithm.

B.1.1.1 BP Graphs

The back-propagation (BP) graphs, e.g. Figure B.5 on page B-7, display the error during the learning phase and is used by all four algorithms. Here, the error (y -axis) is a function of the iteration (x -axis).

B.1.1.2 GA Graphs

The genetic algorithm (GA) graphs (Figure B.3b on page B-5 and Figure B.4a on page B-6) are more complex. The GA graphs are used in the evolution phase of back-propagation/genetic algorithm (BP/GA) and genetic algorithm/back-propagation (GA/BP), and display the minimal, maximal and average fitness, as well as the standard deviation. These are given as functions of the generation (x -axis). All

forms of fitness, both minimal, maximal and average, follow the scale on the left y -axis, while the standard deviation is on the right y -axis.

B.1.2 Testing Graphs

There are two types of representation of the test results: bar view and matrix view. The data are the same, but as the views represent the data in different ways, hopefully most aspects of the results are clear.

B.1.2.1 Bar View

The bar view testing graphs (Figures B.6, B.9, and B.12) display the test results as bars.

As seen in Figure 3.3 on page 35, the output of all the algorithms is a list of numbers — one for each output node (and thus writing system) — which sum to 1. Each of these numbers is the probability that the glyph in question is of the writing system connected to the node of which that number is the output. For each run, the output lists for each glyph are summed up, with regard to the actual writing system. This is then averaged over all the runs. For each of the writing systems along the x -axis, there are ten rectangles indicating each writing system. The height of this rectangle indicates how many of the glyphs from the writing system on the x -axis that were classified as the writing system represented by that rectangle. The maximum height is 1, representing 100%. A perfect classification has thus only ten rectangles, where the rectangle representing the writing system on the x -axis is at 1.0, while the rest are invisible at 0.0.

B.1.2.2 Matrix View

The matrix view testing graphs (Figures B.7, B.10, and B.13) display the test results as a matrix. Each row consists of the test results from the given writing system. The colour of each column in that row indicates which writing systems the glyphs from said writing system were classified as. The brighter the cell, the more glyphs were classified as that writing system. A perfect classification would thus be completely black, except with a white diagonal from top left to bottom right.

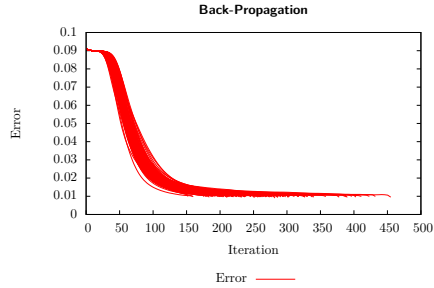
B.2 Main Results

These are the main results achieved by the system, sorted both by the algorithm (Section B.2.1) and by the training/testing phase (Section B.2.2). All available data are used.

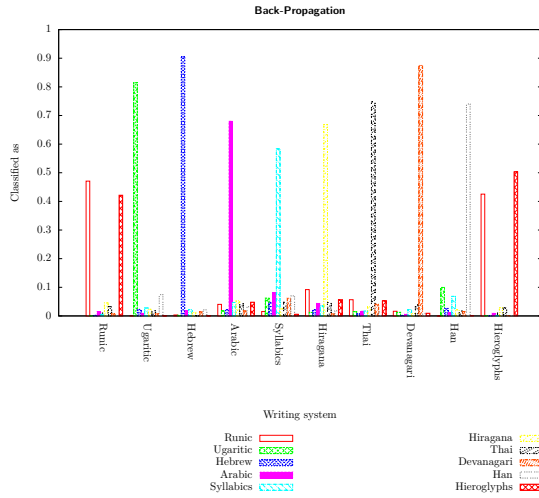
B.2.1 Results Sorted by Algorithm

This section contains all the main results, where the graphs are collected based on the algorithm to which they belong.

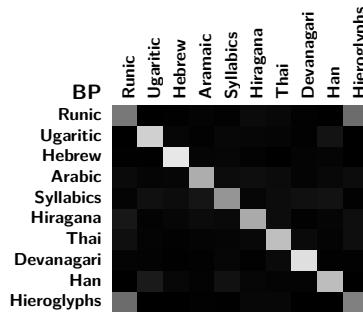
B.2.1.1 Back-Propagation



(a) Training



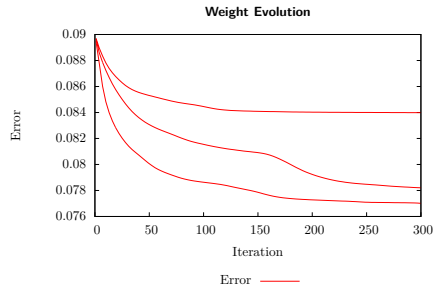
(b) Testing: bar view



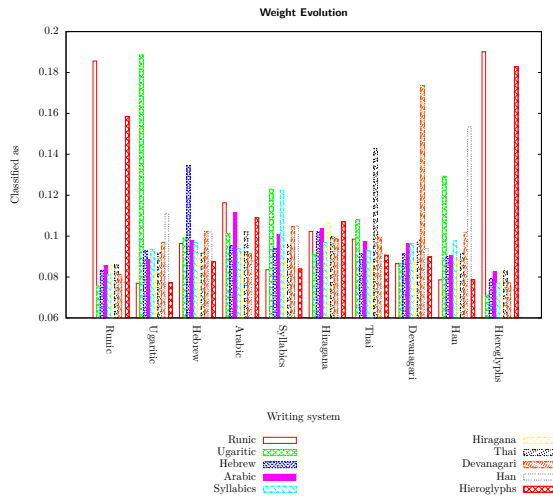
(c) Testing: matrix view

Figure B.1: Training and testing results for BP.

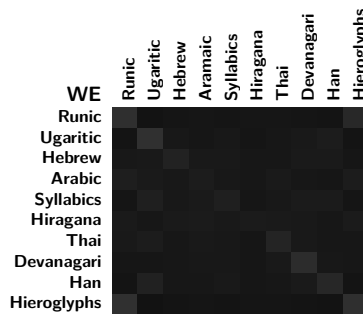
B.2.1.2 Weight Evolution



(a) Training



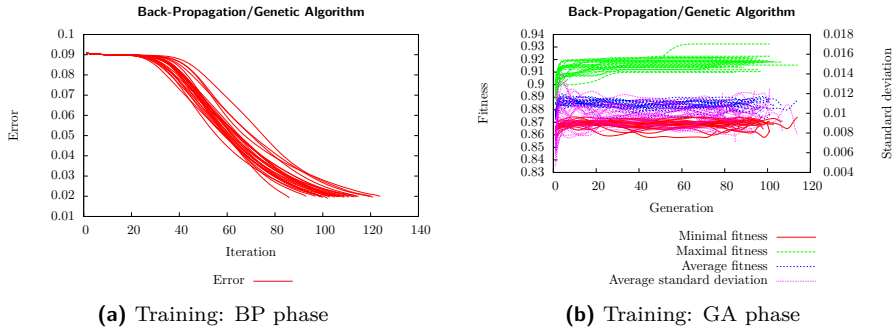
(b) Testing: bar view



(c) Testing: matrix view

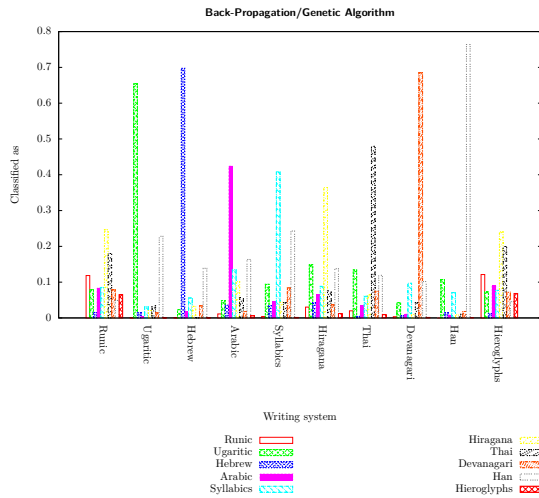
Figure B.2: Training and testing results for WE.

B.2.1.3 Back-Propagation/Genetic Algorithm

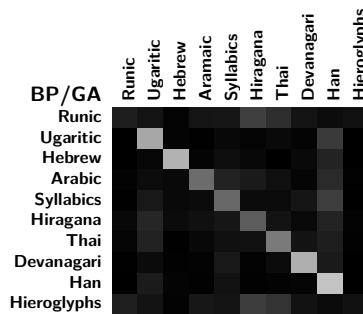


(a) Training: BP phase

(b) Training: GA phase



(c) Testing: bar view



(d) Testing: matrix view

Figure B.3: Training and testing results for BP/GA.

B.2.1.4 Genetic Algorithm/Back-Propagation

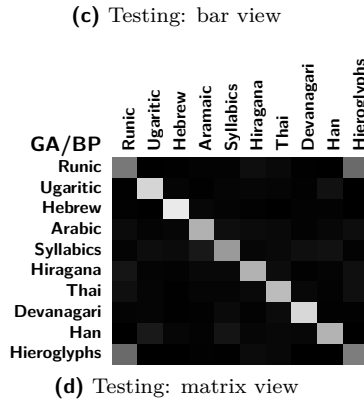
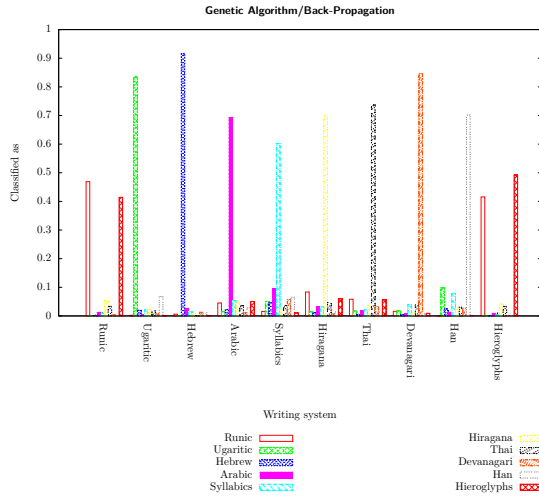
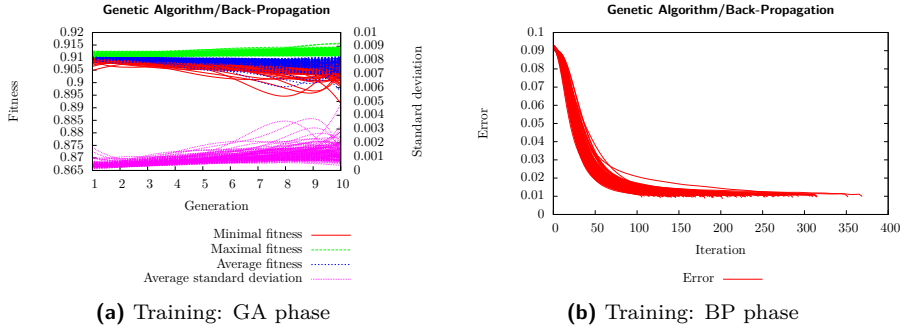


Figure B.4: Training and testing results for GA/BP.

B.2.2 Results Sorted by Phase

This section contains the same graphs as can be found in Section B.2.1, but they are sorted on type instead of algorithm. This is to ease the comparison of the algorithms based on e.g. training and testing results. The training results from the evolutionary phases of BP/GA and GA/BP are not in this collection, but they can be found in Figure B.3b on page B-5 and Figure B.4a on page B-6.

B.2.2.1 Training

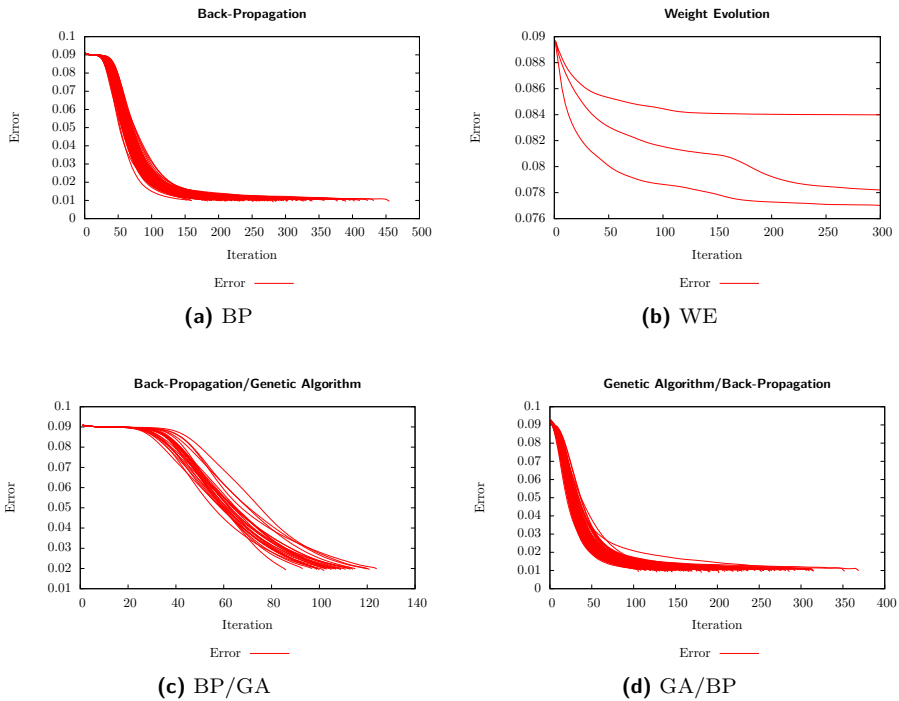


Figure B.5: The training phases with all available data.

B.2.2.2 Testing

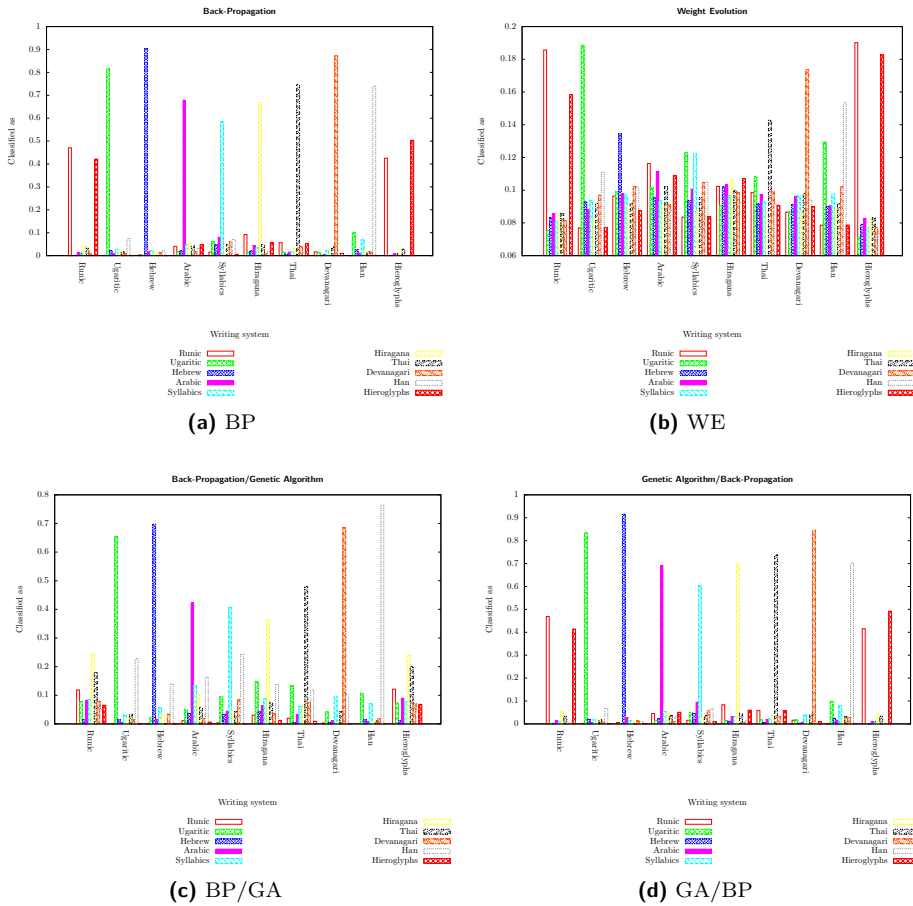


Figure B.6: The testing phases with all available data.

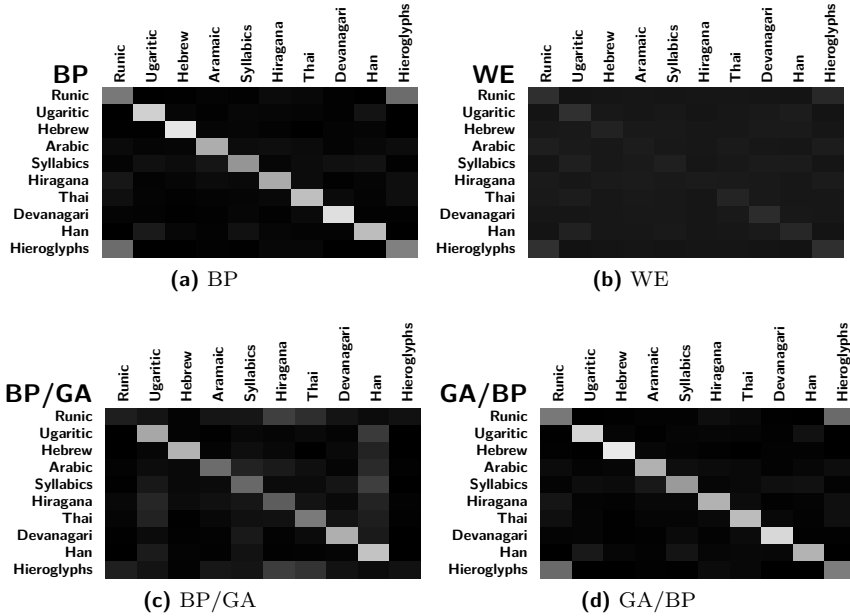


Figure B.7: The testing phases with all available data.

B.3 Different Number of Runs

There are significant differences in the number of runs achieved by the different algorithms, due to the varying runtime. There are 120 runs of BP and GA/BP, 25 of BP/GA, and only 3 of WE. Because of this difference, the graphs were redone using a maximum of n runs. It was tested with both $n = 25$ and $n = 3$. In the former, all algorithms but WE used only the 25 first runs, while WE used the 3 available. In the latter, all algorithms used nothing but the first 3 runs.

This was done to see if the good results of BP and GA/BP were due to the vast number of runs available; since the test results are averaged, minor errors might sort themselves out. By giving all algorithms the same number of runs, this bias would disappear. If BP and GA/BP still are better than WE and BP/GA, it is not because of the larger amount of data available.

B.3.1 Maximum 25 Runs

These graphs are created using the first 25 runs of BP, BP/GA and GA/BP, and the 3 available runs of WE.

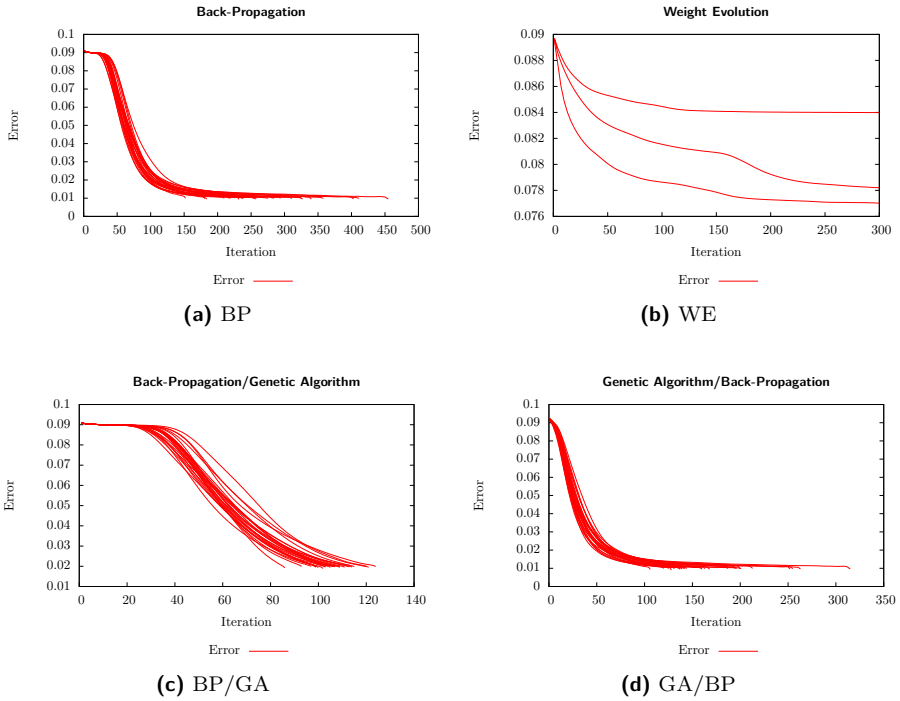


Figure B.8: The training phases with maximum 25 runs.

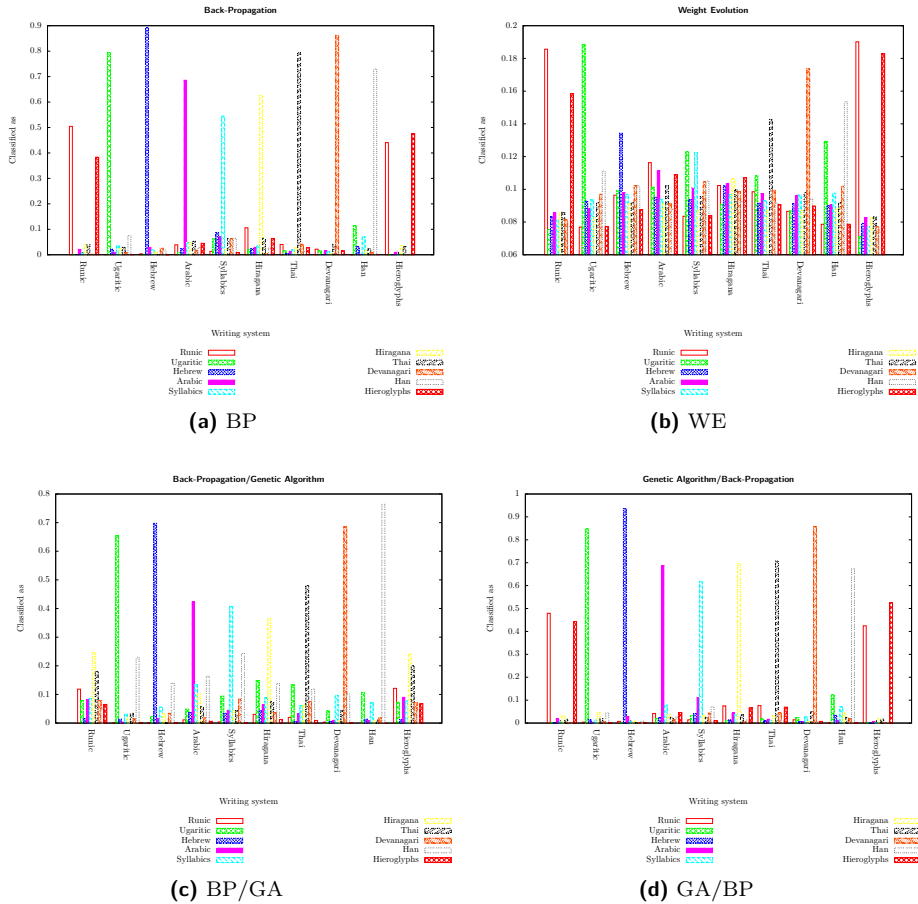


Figure B.9: The testing phases with maximum 25 runs.

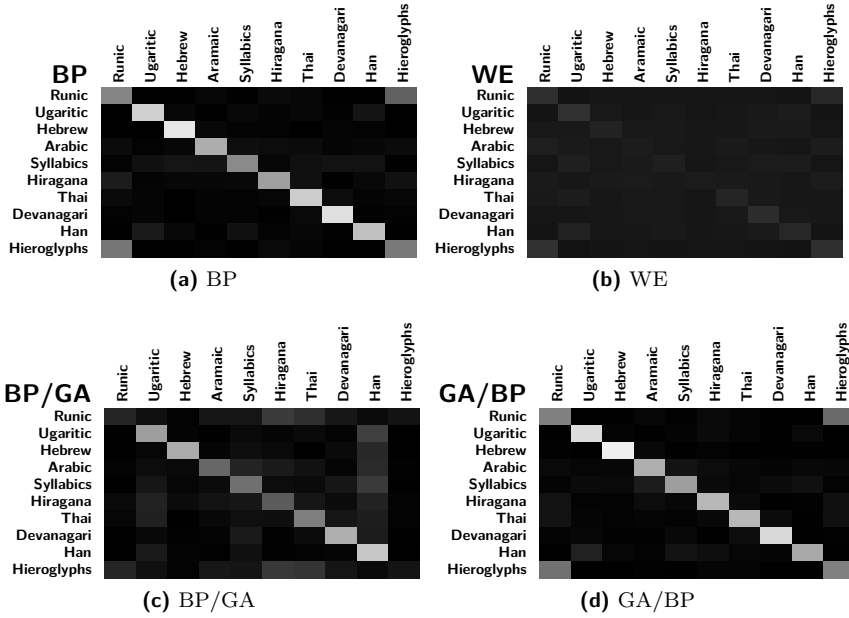


Figure B.10: The testing phases with maximum 25 runs.

B.3.2 Maximum 3 Runs

These results are created using the first 3 runs of all the algorithms.

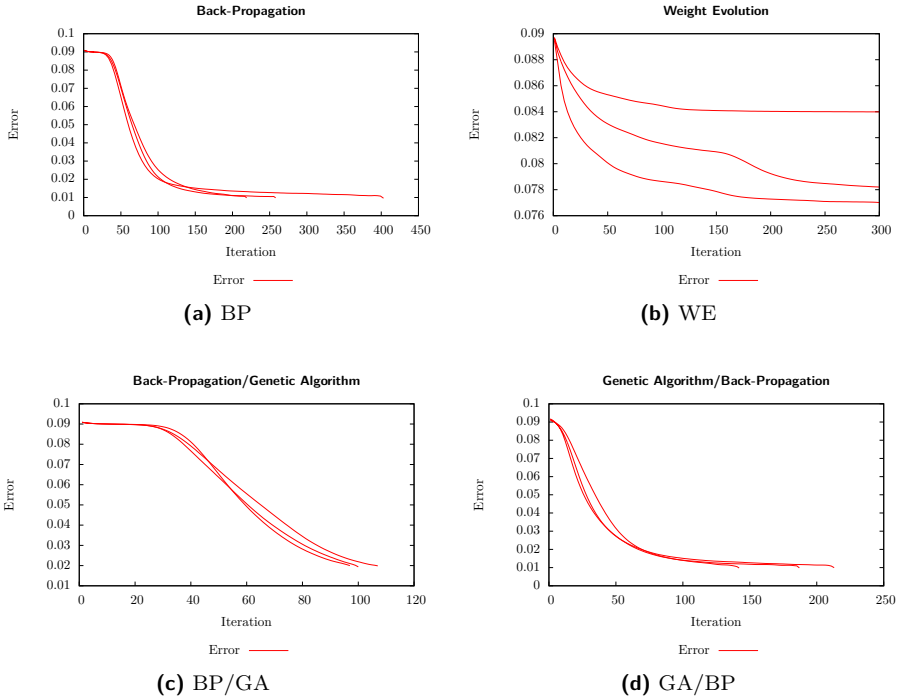


Figure B.11: The training phases with maximum 3 runs.

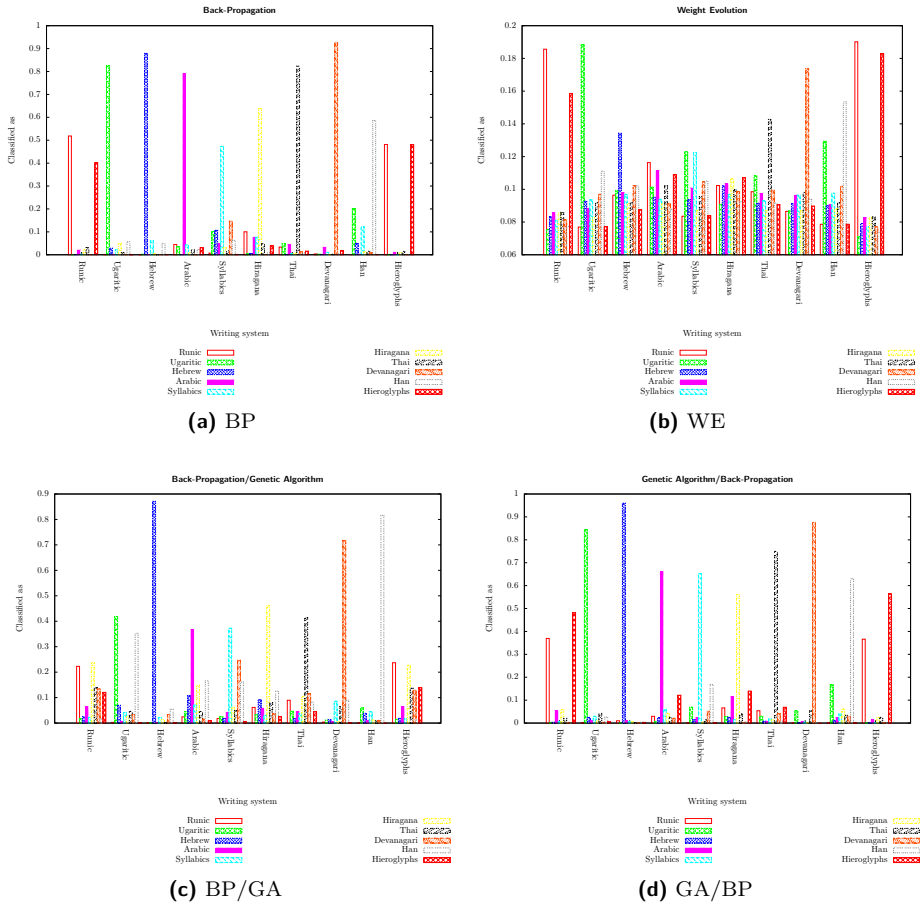


Figure B.12: The testing phases with maximum 3 runs.

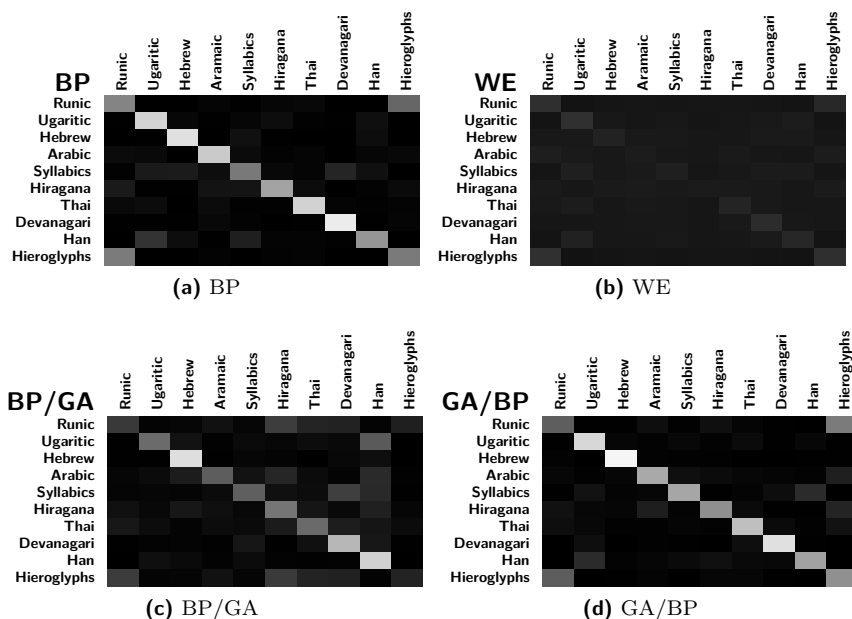


Figure B.13: The testing phases with maximum 3 runs.

B.4 Genetic Algorithm

In this section, the results from the GA by Rødland (2010) are given. This was a pure genetic algorithm, with no learning of any kind.

The GA used also ten writing systems, but not the same as used for the other algorithms; Coptic and Hangul were used instead of Ugaritic and Devanagari. As in the new algorithms, 30 glyphs were used from each writing system, of which 80% were used for training.

Due to the long runtime of the algorithm, only data from four runs were available. Figure B.14a on page B-16 contains the fitness for the four runs, smoothed using a Bezier function. This graph contains both the maximal, minimal and average fitness, as well as the average standard deviation. Figure B.14b on page B-17 contains the test results; the accumulated probability that each of the glyphs is a member of each of the writing systems. Here, the data are averaged over the four runs. Note that the y -axis in Figure B.14b on page B-17 ranges from 0 to 6, representing each of the six glyphs in the test set.

Table B.1: Parameters used by the GA.

Parameter	Value
Crossover rate	0.3
Mutation rate	0.05
Culling	0.2
Elitists	2
Max number of children	60
Number of generations	200
Error threshold	0.1

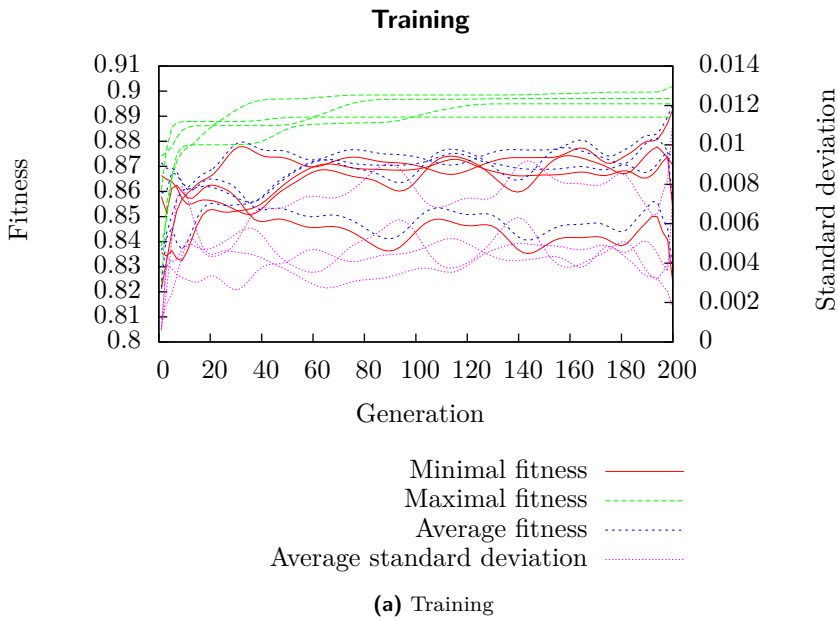
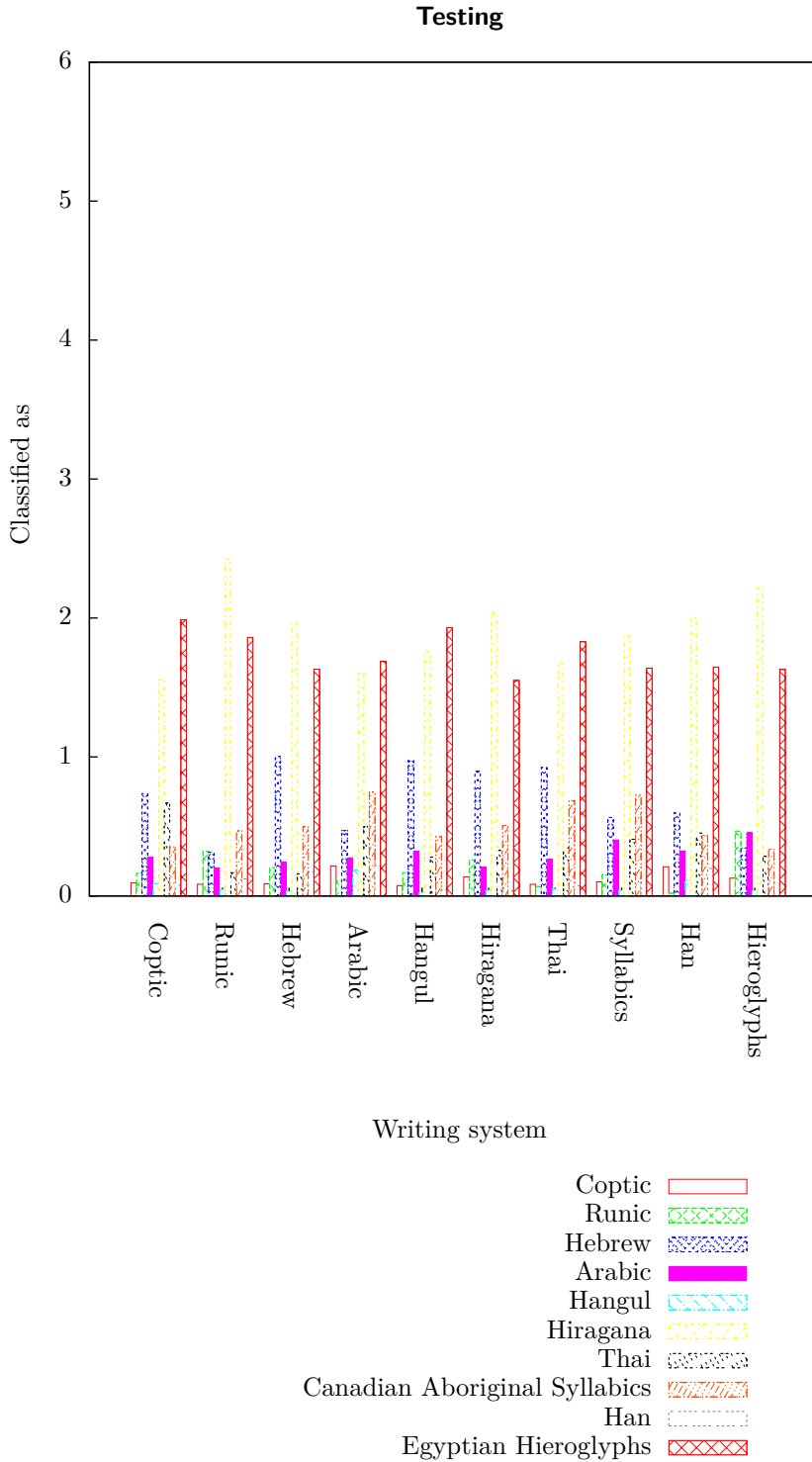


Figure B.14: Training results for the genetic algorithm.



(b) Testing

Figure B.14: (cont.) Testing results for the genetic algorithm.