From Evolving Single Neural Networks to Evolving Ensembles

Xin Yao
School of Computer Science
The University of Birmingham
Edgbaston, Birmingham B15 2TT, UK
Email: x.yao@cs.bham.ac.uk
Phone: +44 121 414 3747, Fax: +44 121 414 4281
URL: http://www.cs.bham.ac.uk/~xin

An Overview of This Talk

1. Integration of Neural and Evolutionary Computation
   (a) Evolving ANN Connection Weights
   (b) Evolving ANN Architectures
   (c) Evolving ANN Learning Rules
   (d) Evolving Modular ANNs
2. Evolving Ensembles and and Automatic Modularisation
3. Ensembles Learning Using Negative Correlation
4. Summary

Introduction

- Learning and evolution are two fundamental forms of adaptation.
- Simulated evolution can be introduced into an ANN at different levels.
- There are many different combinations of neural networks and evolutionary computation techniques.
- This talk will not cover individual neural or evolutionary computation techniques but the *integration* of neural and evolutionary computation techniques.

What Is Evolutionary Computation

1. It is the study of computational systems which use ideas and get inspirations from natural evolution.
2. One of the principles borrowed is *survival of the fittest*.
3. Evolutionary computation (EC) techniques can be used in optimisation, learning and design.
4. EC techniques do not require rich domain knowledge to use. However, domain knowledge can be incorporated into EC techniques.

A Simple Evolutionary Algorithm

1. Generate the initial **population** $P(0)$ at random, and set $i \leftarrow 0$;
2. REPEAT
   (a) Evaluate the fitness of each individual in $P(i)$;
   (b) **Select** parents from $P(i)$ based on their fitness in $P(i)$;
   (c) **Generate** offspring from the parents using *crossover* and *mutation* to form $P(i+1)$;
   (d) $i \leftarrow i+1$;
3. UNTIL halting criteria are satisfied

EA as Population-Based Generate-and-Test

Generate: Mutate and/or recombine individuals in a population.

Test: Select the next generation from the parents and offsprings.

Different Evolutionary Algorithms

There are several well-known EAs with different

- historical backgrounds,
- representations,
- variation operators, and
- selection schemes.

In fact, EAs refer to a whole family of algorithms, not a single algorithm.

Artificial Neural Networks

An ANN can be described as a directed graph in which each node i performs a function, f_i, of the form

$$y_i = f_i\left(\sum_{j=1}^{n} w_{ij}x_j - \theta_i\right) \tag{1}$$

where y_i is the output of the node i, x_j is the jth input to the node, w_{ij} is the connection weight between the node and input x_j, θ_i is the threshold (or bias) of the node, and f_i is the node transfer function.

Learning/Training Algorithms

- Learning process in an ANN is in essence the weight adaptation process. In other words, a training algorithm specifies a procedure of adjusting weights such that the error (as mentioned previously) will be minimised.
- ANN learning can be formulated as a search or optimisation problem.

Major Issues in ANNs

How to improve ANN's generalisation? There are many factors that influence ANN's generalisation:

1. Training algorithm
2. Architecture
3. Implementation
4. ... ...

Integrating Neural and Evolutionary Computation

Many problems in neural networks can be formulated as a search problem, such as

- weight learning,
- architecture adaptation, and
- learning rule adaptation

EAs can deal with these problems more effectively than other techniques.

Evolving Connection Weights

- Learning in ANNs is usually formulated as minimization of an error function, e.g., the total mean square error between target and actual outputs.
- Gradient-based learning algorithms are often used, but
  - they are poor at dealing with multimodal functions and often get trapped in a poor local minimum;
  - they are sensitive to initial conditions;
  - they require the error function to be differentiable;
  - they can be very slow;
  - they may not help improve generalisation.
- EAs are good at dealing with complex, multimodal and nondifferentiable (even discontinuous) functions. They are robust and less sensitive to initial conditions.

Encoding Connection Weights

Before evolution, an encoding scheme (genotypic representation) is often needed to represent ANNs. There are different methods for representing weights.

1. Binary representation
2. Real number representation
3. Hybrid method

The Evolution of Connection Weights

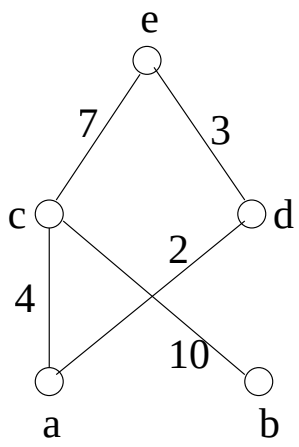
1. Decode each individual (genotype) into a set of connection weights (phenotype).
2. Evaluate the fitness (error) of each individual.
3. Reproduce a number of children for each individual in the current generation according to its fitness.
4. Apply genetic operators, such as crossover and mutation, to each child individual generated above and obtain the next generation.

Assumption: The ANN architecture is fixed during the evolution.

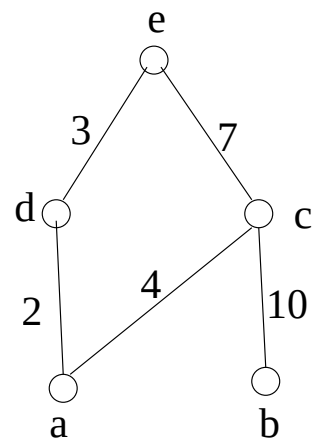
The Permutation Problem

- Also known as the competing convention problem.
- It is caused by the many to one mapping from genotypes to phenotypes since two different genotypes may be mapped to the same phenotype.
- It creates plateau in the search space and makes search inefficient.

Example: Permutation Problem



	a	b	c	d	e
a	0000	0000	0100	0010	0000
b	0000	0000	1010	0000	0000
c	0000	0000	0000	0000	0111
d	0000	0000	0000	0000	0011
e	0000	0000	0000	0000	0000



	a	b	d	c	e
a	0000	0000	0010	0100	0000
b	0000	0000	0000	1010	0000
d	0000	0000	0000	0000	0011
c	0000	0000	0000	0000	0111
e	0000	0000	0000	0000	0000

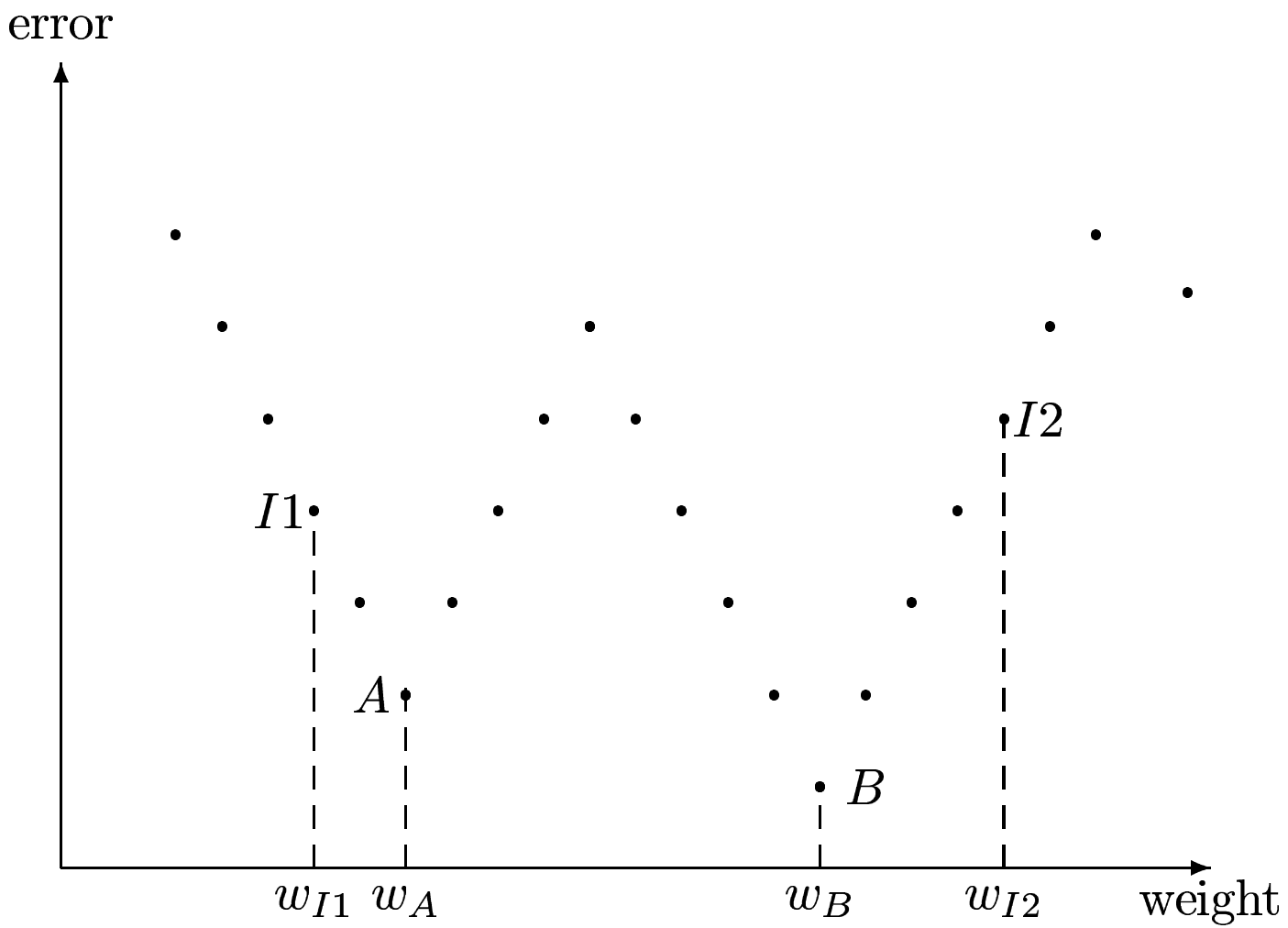
Discussions on Evolutionary Training

1. Evolutionary training is attractive because it can handle a complex, multimodal and nondifferentiable space better. It is particularly appealing when gradient information is unavailable or very costly to obtain or estimate.
2. The evolutionary approach also makes it easier to generate ANNs with some special characteristics.
3. Evolutionary training may be slow for some problems in comparison with fast gradient descent algorithms. However, it always searches for a globally optimal solution.
4. For some problems, evolutionary training is significantly faster and more reliable than gradient descent algorithms.

Hybrid Training

- EAs are not very good at fine-tuned local search although they are good at global search. The efficiency of evolutionary training can be improved significantly by incorporating a local search procedure into the evolution, i.e., combining EA's global search ability with local search's fine-tuning ability.
- One can use EAs to search for a near optimal set of connection weights and then used a local search algorithm to fine tune the weights.

Global Sampling and Local Fine-Tuning



Evolving NN Architectures: Introduction

- Artificial neural networks (ANNs) have mostly been designed manually.
- There are efforts made towards optimising ANN architectures using constructive and pruning algorithms. However, "Such *structural hill climbing* methods are susceptible to becoming trapped at structural local optima." (P. J. Angeline)
- Design of a near optimal ANN architecture can be formulated as a search problem in the architecture space.
- Evolutionary algorithms (EAs) are good candidates for searching this space.

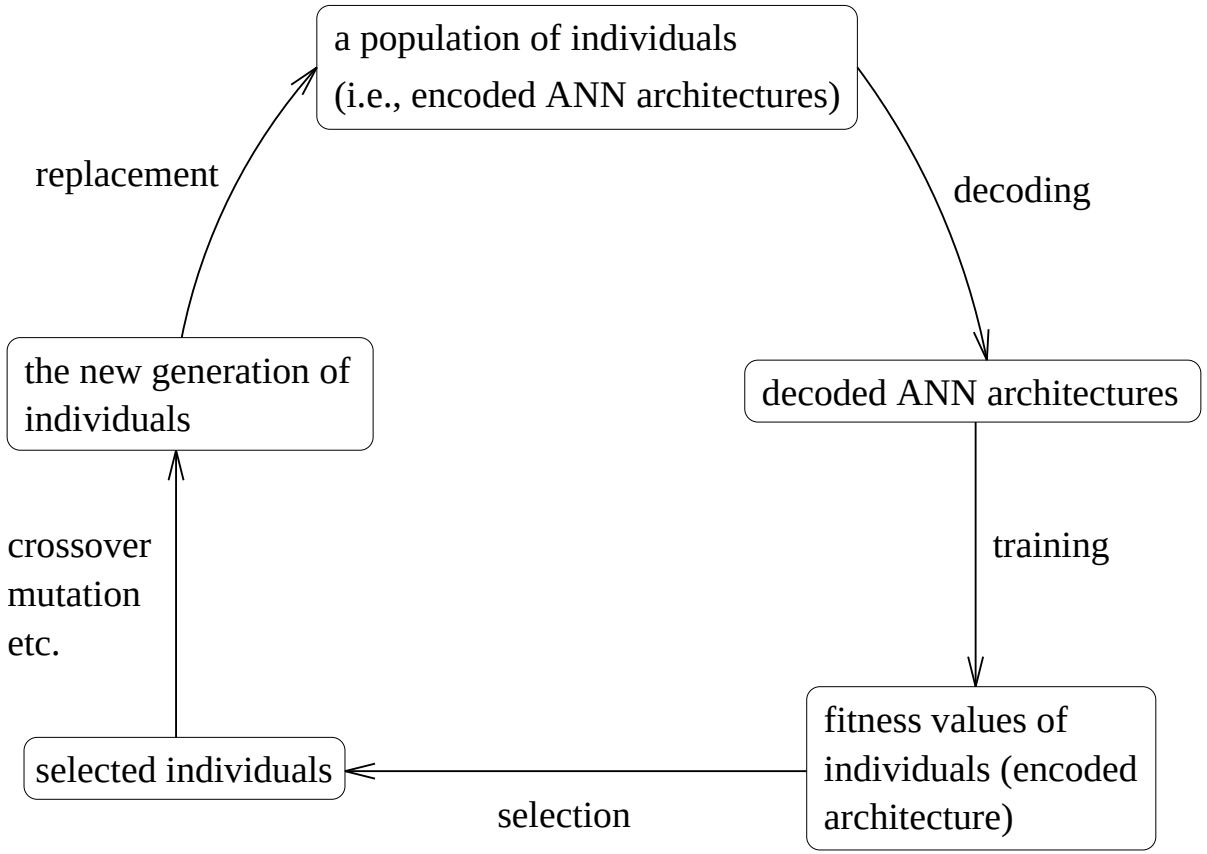
Encoding ANN Architectures

Direct Encoding: All the details, i.e., every connection and node of an architecture are specified by the chromosome.

Indirect Encoding: Only the most important parameters of an architecture, such as the number of hidden layers and hidden nodes in each layer are encoded. Other details about the architecture are left to the training process to decide.

1. Parametric Representation
2. Developmental Rule Representation
3. Others

Evolving NN Architectures: Common Practice



A Typical Cycle of Evolving ANN Architectures

1. Decode each individual in the current generation into an architecture.
2. Train each ANN with the decoded architecture starting from different sets of random initial connection weights.
3. Calculate the fitness of each individual.
4. Reproduce a number of children for each individual in the current generation based on its fitness.
5. Apply variation operators to the children generated above and obtain the next generation.

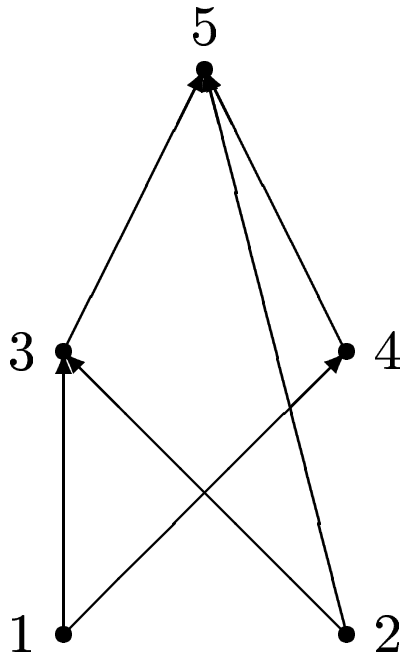
Fitness Evaluation

1. Based on training error
2. Based on validation error
3. Based on training/validation error and network complexity
   (a) based on the number of connections
   (b) based on some information criterion, such as AIC, MDL or MML

The Direct Encoding Scheme

- In the direct encoding scheme, each connection in an architecture is directly specified by its binary representation.
- An $N \times N$ matrix $C = (c_{ij})_{N \times N}$ can represent an architecture with N nodes, where c_{ij} indicates presence or absence of the connection from node i to node j.
- Each such matrix has a direct one-to-one mapping to the corresponding architecture. The binary string representing an architecture is just the concatenation of rows (or columns) of the matrix.

Example 1: Feedforward Networks



(a)

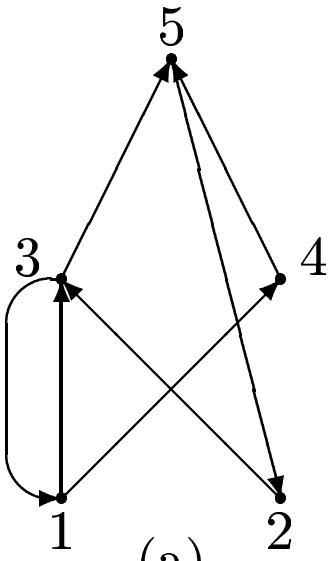
$$\begin{pmatrix} 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

(b)

```
0110 101 01 1
```

(c)

Example 2: Recurrent Networks



(a)

$$\begin{pmatrix} 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

(b)

00110 00100 10001 00001 01000

(c)

Discussions

1. The direct encoding scheme is simple and straightforward to implement.
2. It is suitable for detailed design of architectures.
3. It may facilitate rapid generation and optimization of tightly pruned interesting designs that no one has hit upon so far.
4. It does not scale well for very large architectures.

The Indirect Encoding Scheme

1. Parametric Representation
2. Developmental Rule Representation
3. Other Representations

Parametric Representation

Describe an architecture using a set of parameters, such as the number of hidden layers, the number of hidden nodes in each layer, the number of connections between two layers, etc., without specifying each node-to-node connection.

Advantage: shorter representation

Disadvantage: limited search space

Developmental Rule Representation

- Instead of optimising architectures directly, a set of rules that are used to generate architectures will be optimised.
- This method has several advantages, such as more compact representation and better scalability.
- It also has its own disadvantages though, e.g., noisy fitness evaluation.

Evolving Rules: An Example

- A modified version of graph generation system was used which includes a set of graph generation rules.
- Each rule consists of a left-hand side (LHS) which is a non-terminal symbol and a right-hand side (RHS) which is a 2×2 matrix with either terminal or non-terminal symbols.
- Each rule is represented by five *allele* positions corresponding to five symbols in a rule in a genotype. Each position in a genotype can take the value of an symbol in the range from "A" to "p".
- The 16 rules with "a" to "p" on the left-hand side and 2×2 matrices with only 1's and 0's on the right-hand side are pre-defined and do not participate in evolution.

Example: Rules

$$S \longrightarrow \begin{pmatrix} A & B \\ C & D \end{pmatrix} \quad A \longrightarrow \begin{pmatrix} a & a \\ a & a \end{pmatrix} \quad B \longrightarrow \begin{pmatrix} i & i \\ i & a \end{pmatrix}$$

$$C \longrightarrow \begin{pmatrix} i & a \\ a & c \end{pmatrix} \quad D \longrightarrow \begin{pmatrix} a & e \\ a & e \end{pmatrix} \quad a \longrightarrow \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$$

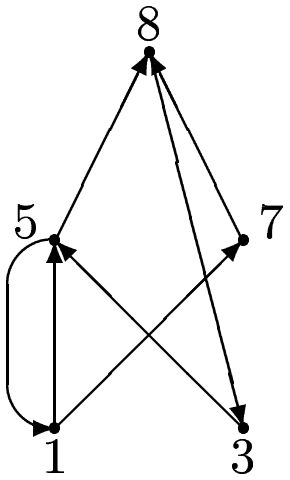
$$c \longrightarrow \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} \quad e \longrightarrow \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \quad i \longrightarrow \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$$

Figure 1: Examples of some developmental rules used to construct a connectivity matrix. S is the initial symbol (or state).

Example: Actual Development

$$S \quad \begin{matrix} A & B \\ C & D \end{matrix} \quad \begin{matrix} a & a & i & i \\ a & a & i & a \\ i & a & a & e \\ a & c & a & e \end{matrix}$$

$$\begin{matrix} 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{matrix}$$



Other Work on Rule Representation

- Mjolsness *et al.* used recursive equations, rather than matrices, to represent rules that specify the growth of connection matrices.
- Gruau's work on cellular encoding.

Other Representations

- Each individual is a hidden node, not an architecture.
- Evolve an architecture layer by layer.
- GAs with fitness sharing is used to evolve feature detectors.

Limitations:

1. Only deals with perceptrons (i.e., +1 or -1 weights)
2. A species may contain more than one individual
3. Limited architectures due to the greedy growing process.

The Evolution of Node Transfer Functions

- The transfer function is often assumed to be the same for all the nodes in an architecture or at least for all the nodes in the same layer.
- Stork *et al.* applied GAs to the evolution of both topological structures and node transfer functions.
- Others have tried to evolve ANN architectures with different nodes, e.g., both sigmoid and Gaussian.

Problem: Noisy Fitness Evaluation

The fitness evaluation of NN architectures is noisy.

1. We want to evaluate architectures without weights (genotypes).
2. We actually evaluate ANN with weights (phenotypes) which
   (a) are initialised at random.
   (b) are trained with a particular algorithm.
3. The evaluation would be even noisier if the developmental rules are not deterministic in the case of an indirect encoding scheme.

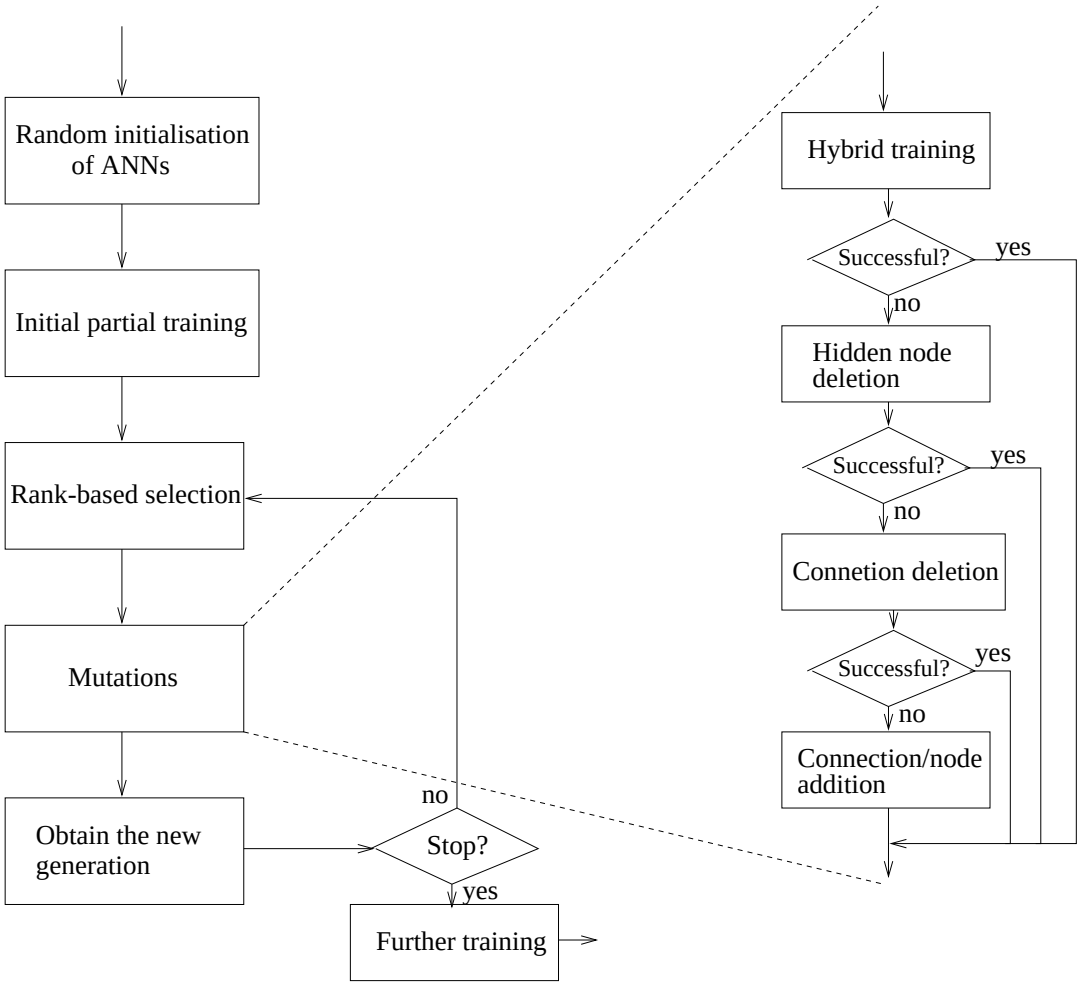
Behaviour Disruption by Genetic Operators

The behavioural difference between the parent and the offspring may be very large after genetic operations. The useful information learned previously may be lost because of such a large disruption to the NN behaviour.

Our Approach

1. Simultaneous evolution of both architectures and weights.
2. Emphasise behavioural rather than genetic evolution (e.g., partial training after mutation, node split, etc.)
3. Use mutation only
4. Use bias rather than a complexity (regularisation) term to encourage compact NNs that generalise well.

An Evolutionary System — EPNet



Mutation

1. Five mutation operators are used in our system: partial training, node deletion, connection deletion, connection addition, and node addition.
2. They are applied sequentially. Only one of them will be used in each generation.
3. Partial training is the only mutation operator used to change NN's connection weights. It uses a hybrid algorithm to train the NN for a *fixed* number of epochs. Such training does not guarantee NN's convergence. Hence the training is *partial*.

Connection Deletion

Connections are probabilistically selected for deletion according to

$$test(w_{ij}) = \frac{\sum_{t=1}^{T} \xi_{ij}^t}{\sqrt{\sum_{t=1}^{T} (\xi_{ij}^t - \overline{\xi}_{ij})}} \tag{2}$$

where $\xi_{ij}^t = w_{ij} + \Delta w_{ij}^t(w)$, $\overline{\xi}_{ij}$ denotes the average over the set ξ_{ij}^t, $t = 1, \ldots, T$, $\Delta w_{ij}(w) = -\eta[\partial L_t / \partial w_{ij}]$ is the weight update, L ($L = \sum_{t=1}^{T} \sum_{i=1}^{n} |Y_i(t) - Z_i(t)|$) is a linear error function with respect to example t and weight w_{ji}.

Connection and Node Addition

1. Connections are probabilistically added according to Eq.(2). They are selected from all connections with zero weight.
2. Hidden nodes are added through node splitting. Two nodes obtained by splitting an existing node i have the same connections as the existing node. The weights of these new nodes have the following values:

$$w_{ij}^1 = w_{ij}^2 = w_{ij},\ i \geq j,$$

$$w_{ij}^1 = (1+\alpha)w_{ij}, w_{ij}^2 = -\alpha w_{ij},\ i < j$$

where $\mathbf{w}$, $\mathbf{w}^1$ and$\mathbf{w}^2$ are the weight vectors, and α is a mutation parameter.

Experimental Studies

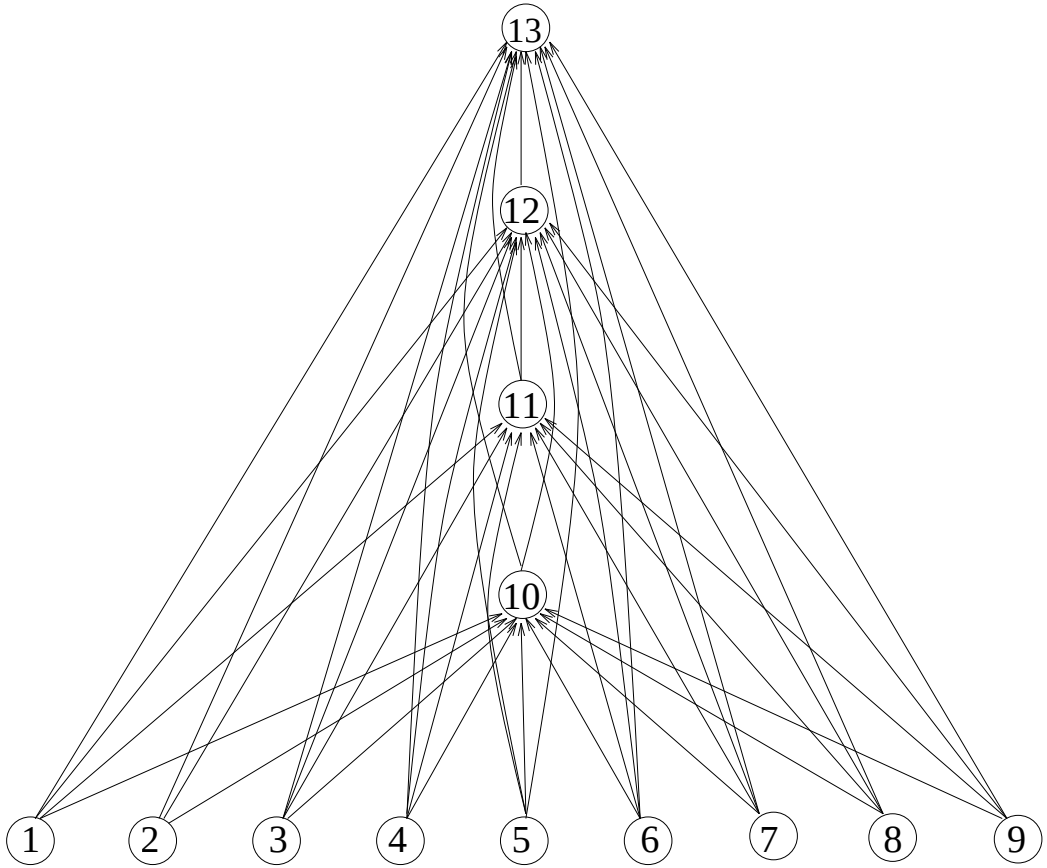
- Parity problems of sizes between 4 and 9.
- The two-spiral problem.
- Medical diagnosis; heart disease, breast cancer, diabetes, thyroid.
- Australian credit card problem
- Time-series predictions

Comparison: Number of Hidden Nodes

N	EPNet	CCA	PCA	TA
4	2	2	2	2
5	2	2	2	2
6	2	3	3	3
7	2	4	3	3
8	3	5	4	4

Table 1: Number of hidden nodes in an ANN for the N-parity problem. EPNet — our algorithm, CCA — Cascade-Correlation Algorithm, PCA — Perceptron Cascade Algorithm, TA — Tower Algorithm.

The 9-Parity Problem — Architectures



The 9-Parity Problem — Weights and Biases

	T	1	2	3	4	5	6
10	-12.35	-12.19	12.38	-12.19	-12.20	12.23	-12.19
11	-4.12	6.04	0	6.04	6.04	-6.34	6.04
12	7.05	6.78	-7.13	6.78	6.79	-6.96	6.79
13	0	7.89	-7.76	7.89	7.89	-8.04	7.89

	7	8	9	10	11	12
10	12.23	-12.12	-12.20	0	0	0
11	-6.34	-26.16	6.05	0	0	0
12	-6.96	-9.59	6.79	26.70	-16.45	0
13	-8.04	-10.71	7.89	30.71	-18.99	-17.02

Australian Credit Card Problem

- There are 690 cases in total. The output has two classes. The 14 attributes include 6 numeric values and 8 discrete ones, the latter having from 2 to 14 possible values.
- The whole data set is randomly partitioned into (1) a training subset (346 cases), (2) V-set 1 (86 cases), (3) V-set 2 (86 cases), and (4) a testing subset (172 cases).

Experimental Results: Architectures

Averaged over 30 runs.

	Mean	Std Dev	Min	Max
Number of connections	88.03	24.70	43	127
Number of hidden nodes	4.83	1.62	2	7

Experimental Results: Error Rates

Averaged over 30 runs.

	Training set Error rate	V-set 1 Error rate	V-set 2 Error rate	Test set Error rate
Mean	0.111	0.074	0.091	0.115
SD	0.021	0.022	0.024	0.019
Min	0.081	0.035	0.047	0.081
Max	0.173	0.105	0.140	0.157

Experimental Results: Comparisons

	EPNet	Cal5	ITrule	DIPOL92
error rate	0.115	0.131	0.137	0.141
	Discrim	Logdisc	CART	RBF
error rate	0.141	0.141	0.145	0.145
	CASTLE	NaiveBay	IndCART	BP
error rate	0.148	0.151	0.152	0.154

Time Series Prediction: Mackey-Glass Problem

Method	Number of Connections	Testing Error $\Delta t = 6$	Testing Error $\Delta t = 84$
EPNet	103	0.02	0.06
BP	540	0.02	0.05
CC Learning	693	0.06	0.32

Table 2: Generalisation results comparison among EPNet, BP, and CC learning for the Mackey-Glass time-series prediction problem.

Time Series Prediction: Logistic Map Problem

Method	Number of Connections	Testing Error
EPNet	33.03	0.0257
ERP	-	0.0566
1-10-1 BP network	20	0.1145
1-15-1 BP network	30	0.0387

Table 3: Comparison among EPNet, ERP, and 1-N-1 feedforward networks for the logistic map time-series prediction problem. '-' in the table means 'not available'.

What Have We Been Doing So Far

- Learning? Evolving? Training? Designing?
- Minimising the error!
- What error?
- Training error!

Learning and Optimisation

- Learning has often been formulated as an optimisation problem.
- However, learning is different from optimisation *in practice*.

1. In optimisation, the fitness function reflects what is needed. The optimal value is always better than the second optimal one.
2. In learning, there is no way to quantify generalisation exactly. A system with minimum training error may not be the one with the best generalisation ability.
3. *Why select the "best" individual in a population if we are not clear what "best" means?*

Exploit Useful Information in a Population

- Since an individual with the minimum training error may not be the one with best generalisation, it makes sense to exploit useful information in a population rather than any single individual.
- All in all, it is a whole *population* that is evolving, not a single individual.

A Population of Heads Are Better Than One

Instead of using the best individual and discard the rest of the population, individuals in a population can be combined to form an ensemble system.

Experiment 1: Evolving a population of ANNs.

Experiment 2: Evolving a population of rule sets.

These experiments compare the best individual against a combination of all individuals from the population.

Experiment 1: EPNet Continued

Data set	Method	Testing Error Rate
Card	EPNet	0.100
	Ensemble	0.093
Diabetes	EPNet	0.232
	Ensemble	0.226
Heart	EPNet	0.154
	Ensemble	0.151

Designing Better Heads

- Speciation by fitness sharing is one of the techniques which encourage *automatic* formation of species. Different species are good at different things.
- Species are modules of a larger integrated system.

Speciation by Implicit Fitness Sharing

For each strategy i in the GA population, do the following C times:

1. From the GA population, select a sample of σ strategies.
2. Find the strategy in that sample which achieves the highest score (or the largest winning margin, if you prefer) against the single test strategy i.
3. The best in the sample receives payoff. In the case of a tie, payoff is shared equally among the tie-breakers.

Figure 2: Payoff function for implicit fitness sharing.

Experiment 2: Evolving Game-Playing Strategies

1. A genetic algorithm was used to evolve strategies for playing the 2IPD.
2. Implicit fitness sharing was used to form different species (specialists).
3. A gating algorithm was used to decide which species should respond to an unknown opponent.
4. Experimental results showed that our method worked very well.

A Combination Method — the Gating Algorithm

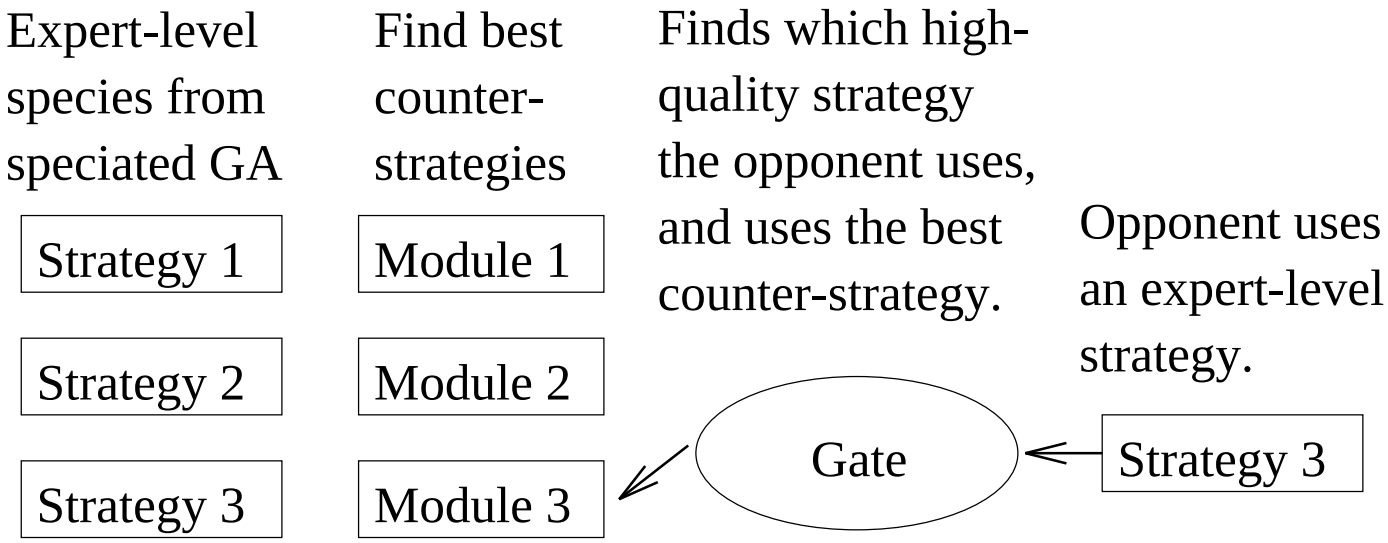


Figure 3: A gating algorithm for combining different expertise in a population together.

Experimental Results

	$l = 4$			
Strategy	Wins	Ties	Average Score	
	%	%	Own	Other's
best.sr	0.360	0.059	1.322	1.513
gate.sr	0.643	0.059	1.520	1.234

Table 4: Results against the best 25 strategies from the partial enumerative search, for 2IPD with remembered history $l = 4$. The results were averaged over 30 runs.

Negatively Correlated Learning

The idea of designing different cooperative specialists is *not* limited to EAs. It can be used by gradient descent algorithms too.

1. Making individuals different:

$$E_i = \frac{1}{N}\sum_{n=1}^{N}\left(\frac{1}{2}(F_i(n) - d(n))^2 + \lambda p_i(n)\right)$$

where

$$p_i(n) = (F_i(n) - F(n))\sum_{j\neq i}(F_j(n) - F(n))$$

$F(n)$ is the ensemble output.

2. All individuals are learned *simultaneously*.

Ensemble Network Learning

We consider estimating $g(\mathbf{x}) = E[d|\mathbf{x}]$ by forming a simple average of a set of outputs of individual networks which are trained using the same training data set D

$$F(\mathbf{x}, D) = \frac{1}{M}\Sigma_{i=1}^{M} F_i(\mathbf{x}, D) \tag{3}$$

where $F_i(\mathbf{x}, D)$ is the actual response of network i and M is the number of neural network estimators.

Bias-Variance-Covariance Trade-off

Taking expectations with respect to the training set D, the expected mean-squared error of the combined system can be written in terms of individual network output

$$E_D\left[\left(E[d|x]-F(\mathbf{x},D)\right)^2\right]=\left(E_D[F(\mathbf{x},D)]-E[d|x]\right)^2$$
$$+E_D\left[\frac{1}{M^2}\Sigma_{i=1}^{M}\left(F_i(\mathbf{x},D)-E_D[F_i(\mathbf{x},D)]\right)^2\right]$$
$$+E_D\left[\frac{1}{M^2}\Sigma_{i=1}^{M}\Sigma_{j\neq i}\left(F_i(\mathbf{x},D)-E_D[F_i(\mathbf{x},D)]\right)\right.$$
$$\left.\left(F_j(\mathbf{x},D)-E_D[F_j(\mathbf{x},D)]\right)\right] \quad (4)$$

The expectation operator E_D represents the average over all the patterns in the training set D.

How to Choose the Correlation Penalty

The purpose of minimising p_i is to negatively correlate each individual's error with errors for the rest of the ensemble.

- The function p_i for regression problems can be chosen as

$$p_i(n) = (F_i(n) - d(n)) \, \Sigma_{j \neq i} \, (F_j(n) - d(n)) \tag{5}$$

  for noise free data, or

$$p_i(n) = (F_i(n) - F(n)) \, \Sigma_{j \neq i} \, (F_j(n) - F(n)) \tag{6}$$

  for noisy data.

- The function p_i for classification problem can be chosen as

$$p_i(n) = (F_i(n) - 0.5) \, \Sigma_{j \neq i} \, (F_j(n) - 0.5) \tag{7}$$

Experimental Studies

- The Australian credit card assessment problem was used.
- The whole data set is randomly partitioned into a training (518 cases) and a testing set (172 cases).
- The ensemble used in our experiment consisted of five strictly-layered feedforward neural networks. All individual networks had the same architecture. They had three layers and 5 hidden nodes in the hidden layer. The learning-rate η in BP is 0.1, and λ is 0.375. These parameters were chosen after limited preliminary experiments. They are not meant to be optimal.

Experiment Results: Error Rates

	Training set		Test set	
# epochs	500	1000	500	1000
Mean	0.1093	0.0846	0.1177	0.1163
SD	0.0092	0.0088	0.0182	0.0159
Min	0.0927	0.0676	0.0698	0.0756
Max	0.1255	0.1004	0.1628	0.1454

Table 5: Error rates for the Australian credit card assessment problem. The results were averaged over 25 runs.

Experiment Results: Evolutionary Process

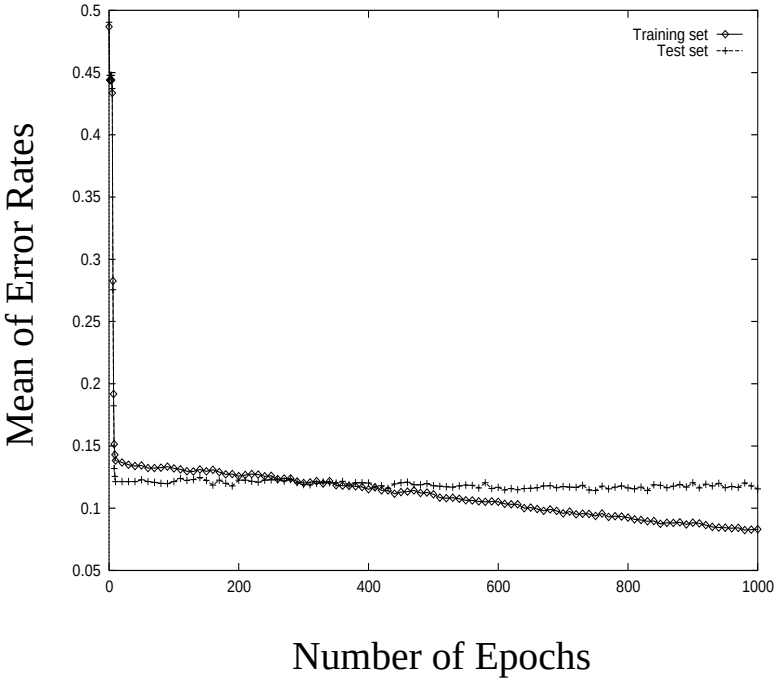


Figure 4: The average result over 25 runs of our ensemble learning algorithm. The horizontal axis indicates the number of training epochs. The vertical axis indicates the error rate.

Experiment Results: Being Different

$\Omega_1 = 146$	$\Omega_2 = 148$	$\Omega_3 = 149$	$\Omega_4 = 151$
$\Omega_{12} = 137$	$\Omega_{13} = 137$	$\Omega_{14} = 141$	$\Omega_{23} = 140$
$\Omega_{24} = 141$	$\Omega_{34} = 139$	$\Omega_{123} = 132$	$\Omega_{124} = 133$
$\Omega_{134} = 133$	$\Omega_{234} = 134$	$\Omega_{1234} = 128$	

Table 6: The sizes of the correct response sets of individual networks on the testing set and their intersections for the Australian credit card assessment problem.

Comparison with Other Work

Algorithm	TER	Algorithm	TER
NCNN	0.116	Logdisc	0.141
EPNet	0.115	CART	0.145
Evo-En-RLS	0.131	RBF	0.145
Cal5	0.137	CASTLE	0.148
ITrule	0.141	NaiveBay	0.151
DIPOL92	0.141	IndCART	0.152

Table 7: Comparison among the negative correlation NN (NCNN), EPNet, an evolutionary ensemble learning algorithm (Evo-En-RLS), and others in terms of the average testing error rate. TER stands for Testing Error Rate in the table.

Mackey-Glass Time Series Prediction Problem

Method	Testing RMS	
	$\Delta t = 6$	$\Delta t = 84$
NCNN	0.01	0.03
EPNet	0.02	0.06
BP	0.02	0.05
CC Learning	0.06	0.32

Table 8: The "Testing RMS" in the table refers to the normalised root-mean-square error on the testing set.

Comparision between NCNN and ME

Adding noises.

Method	E_{mse}	
	$\sigma^2 = 0.1$	$\sigma^2 = 0.2$
NCNN	0.012	0.023
ME	0.018	0.038

Table 9: Comparison between NCNN and the mixtures-of-experts (ME) architectures in terms of the integrated mean-squared error on the testing set for the moderate noise case and the large noise case.

Evolving ANN Ensembles

No need to predefine the number of ANNs in an ensemble.

1. Generate an initial population of M NNs, and set $k = 1$.
2. Train each NN in the population on the training set for a certain number of epochs using the negative correlation learning.
3. Randomly choose n_b NNs as parents to create n_b offspring NNs by Gaussian mutation.
4. Add the n_b offspring NNs to the population and train the offspring NNs using the negative correlation learning while the rest NNs' weights are frozen.
5. Calculate the fitness of each NN in the population and prune the population to the M fittest NNs.
6. Stop if certain criteria are satisfied and go to Step 7. Otherwise, $k = k + 1$ and go to Step 3.
7. Cluster the NNs in the population. These clusters are then used to construct NN ensembles.

Fitness Sharing and Fitness Evaluation

- An implicit fitness sharing is used based on the idea of "covering" the same training case by shared individuals. The procedure of calculating shared fitness is carried out case-by-case over the training set.
- For each training case, if there are $p > 0$ individuals that correctly classify it, then each of these p individuals receives $1/p$ fitness reward, and the rest individuals in the population receive zero fitness reward. The fitness reward is summed over all training cases. This method is expected to generate a smoother shared fitness landscape.

Evolving Connection Weights

- Learning in ANNs is usually formulated as minimization of an error function, e.g., the total mean square error between target and actual outputs.
- Gradient-based learning algorithms are often used, but
  - they are poor at dealing with multimodal functions and often get trapped in a poor local minimum;
  - they are sensitive to initial conditions;
  - they require the error function to be differentiable;
  - they can be very slow;
  - they may not help improve generalisation.
- EAs are good at dealing with complex, multimodal and nondifferentiable (even discontinuous) functions. They are robust and less sensitive to initial conditions.

Evolving Learning Rules

- Learning rules are seldom touched in neural network learning. Most people use the Hebbian or generalised delta rule.
- Different architectures may have different best learning rules for them.
- Adaptation of a learning rule can be regarded as the first step towards learning to learn.

How to Evolve Learning Rules

1. Decode each individual in the current generation into a learning rule.
2. Construct a set of ANNs with randomly generated architectures (or a pre-defined architecture in some cases) and initial connection weights and train them using the decoded learning rule.
3. Calculate the fitness of each individual.
4. Reproduce a number of intermediate children.
5. Apply search operators to the child individuals produced above and obtain the new generation.

The Evolution of Learning Rules

- Unlike the evolution of connection weights and architectures which only deal with static objects in an ANN, i.e., weights and architectures, the evolution of learning rules has to work on the dynamic behaviour of an ANN.
- Trying to develop a universal representation scheme which can specify any kind of dynamic behaviours is clearly impractical.

Two Assumptions

1. The weight-updating of a connection depends only on local information such as the activation of the input node, the activation of the output node, the current connection weight, etc.
2. The learning rule is the same for all connections in an ANN.

A Common Form of Learning Rule

A learning rule is assumed to be a linear function of these local variables and their products. That is, a learning rule can be described by the function

$$\Delta w(t) = \sum_{k=1}^{n} \sum_{i_1, i_2, \cdots, i_k = 1}^{n} \left(\theta_{i_1 i_2 \cdots i_k} \prod_{j=1}^{k} x_{i_j}(t-1) \right) \qquad (8)$$

where t is time, Δw is the weight change, $x_1, x_2, \cdots, x_n$ are local variables and θ's are real-valued coefficients which are decided by the evolution of learning rules.

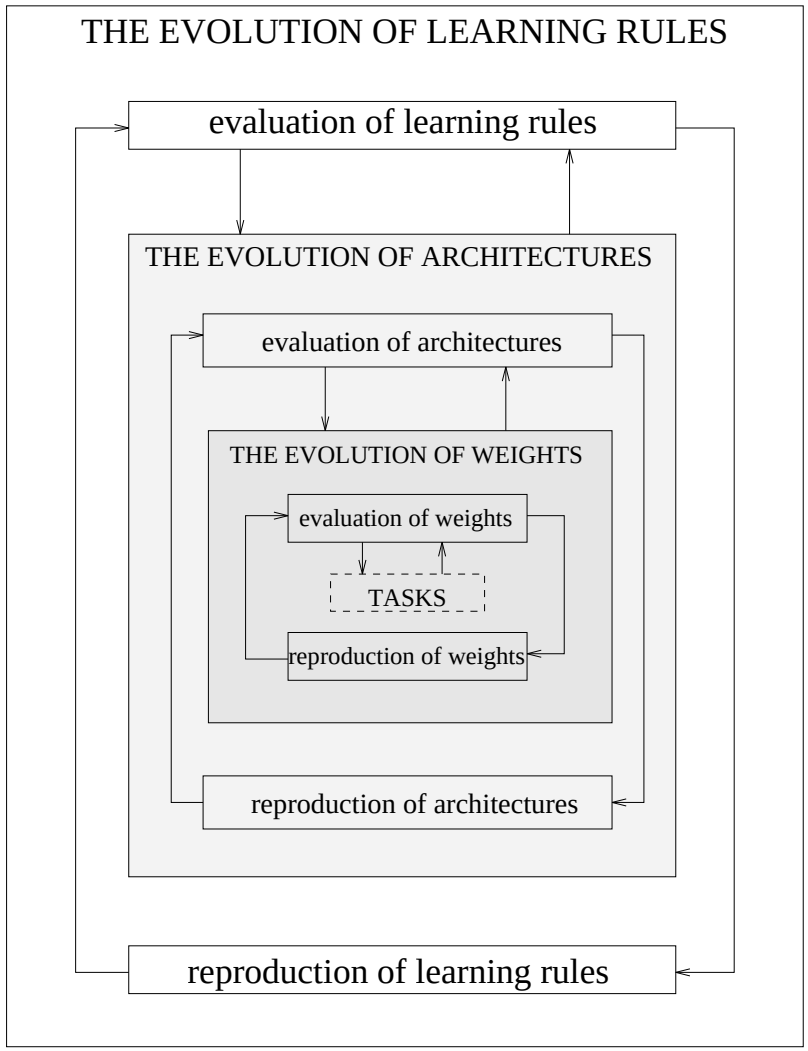
Three Basic Issues

1. determination of a subset of terms described in Eq.8
2. representation of their coefficients as genotypes
3. the evolution simulated by an evolutionary algorithm

Chalmers' Experiments

1. A learning rule is a linear combination of four local variables and their six pairwise products. No third or fourth order terms were used.
2. Ten coefficients and a scale parameter were encoded in a binary string via exponential encoding.
3. The architecture used in the fitness evaluation was fixed because only single layer ANNs were considered and the number of inputs and outputs were fixed by the learning task at hand.
4. The evolution was able to discover the well-known delta rule and some of its variants.

A Framework for Adaptive Systems



Summary

- Simulated evolution can produce very compact NNs that generalise well.
- Population information should be and can be exploited in evolutionary learning.
- Both fitness sharing and negative correlation encourage a population of cooperating species.

References (Downloadable)

1. Y. Liu, X. Yao and T. Higuchi, "Evolutionary Ensembles with Negative Correlation Learning," *IEEE Transactions on Evolutionary Computation*, **4**(4):380-387, November 2000.
2. X. Yao, "Evolving artificial neural networks," *Proceedings of the IEEE*, **87**(9):1423-1447, September 1999.
3. Y. Liu and X. Yao, "Simultaneous training of negatively correlated neural networks in an ensemble," *IEEE Trans. on Systems, Man, and Cybernetics, Part B: Cybernetics*, **29**(6):716-725, December 1999.
4. X. Yao, Y. Liu and G. Lin, "Evolutionary programming made faster," *IEEE Trans. on Evolutionary Computation*, **3**(2):82-102, July 1999.
5. X. Yao and Y. Liu, "Making use of population information in evolutionary artificial neural networks," *IEEE Trans. on Systems, Man, and Cybernetics, Part B: Cybernetics*, **28**(3):417-425, June 1998.
6. X. Yao and Y. Liu, "A new evolutionary system for evolving artificial neural networks," *IEEE Trans. on Neural Networks*, **8**(3):694-713, May 1997.
7. P. Darwen and X. Yao, "Speciation as automatic categorical modularization," *IEEE Trans. on Evolutionary Computation*, **1**(2):101-108, 1997.