

Lecture 18: Evolving Learning Machines (Part II)

1. Review of the last lecture
 - (a) Two major approaches to evolutionary learning
 - (b) Evolving rule-based systems
2. This lecture: Evolving artificial neural networks
3. 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.

Integrating Neural and Evolutionary Computation

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

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

Evolutionary algorithms (EAs) can deal with these problems more effectively than other techniques.

ANN Learning

- 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 (such as BP) 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

The Evolution of Connection Weights

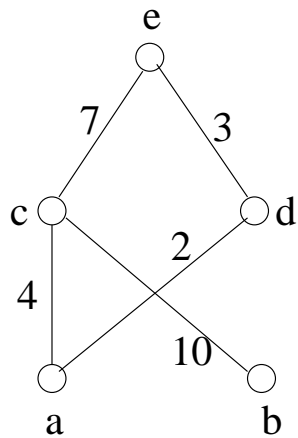
1. Decode each individual (genotype) into a set of connection weights (phenotype).
2. Evaluate the fitness (error) of each individual.
3. Reproduce a number of children for each individual in the current generation according to its fitness.
4. Apply evolutionary operators, such as crossover and mutation,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

Assumption: The ANN architecture is fixed during the evolution.

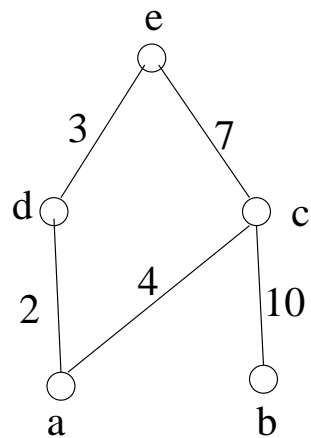
The Permutation Problem

- 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.

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

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.

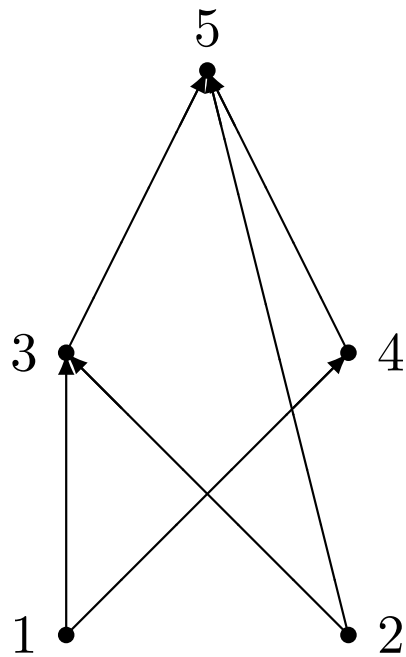
Fitness Evaluation

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)

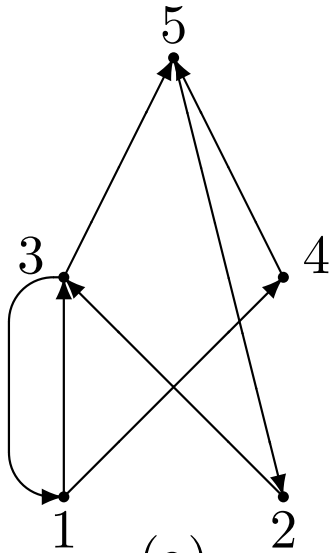
$$\begin{pmatrix} 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

(b)

0110 101 01 1

(c)

Example 2: Recurrent Networks



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

(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

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 a 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

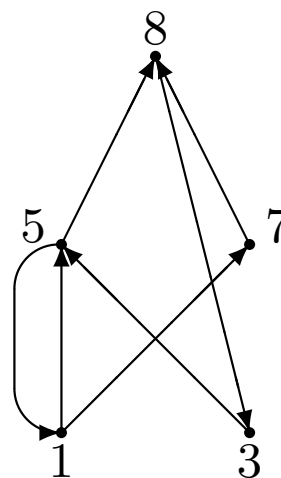
$$\begin{array}{lll}
 S \longrightarrow \begin{pmatrix} A & B \\ C & D \end{pmatrix} & A \longrightarrow \begin{pmatrix} a & a \\ a & a \end{pmatrix} & B \longrightarrow \begin{pmatrix} i & i \\ i & a \end{pmatrix} \\
 C \longrightarrow \begin{pmatrix} i & a \\ a & c \end{pmatrix} & D \longrightarrow \begin{pmatrix} a & e \\ a & e \end{pmatrix} & a \longrightarrow \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \\
 c \longrightarrow \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} & e \longrightarrow \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} & i \longrightarrow \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}
 \end{array}$$

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

Example: Actual Development

					<i>a</i>	<i>a</i>	<i>i</i>	<i>i</i>
<i>S</i>	<i>A</i>	<i>B</i>			<i>a</i>	<i>a</i>	<i>i</i>	<i>a</i>
	<i>C</i>	<i>D</i>			<i>i</i>	<i>a</i>	<i>a</i>	<i>e</i>
					<i>a</i>	<i>c</i>	<i>a</i>	<i>e</i>

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



Problem 1: 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.

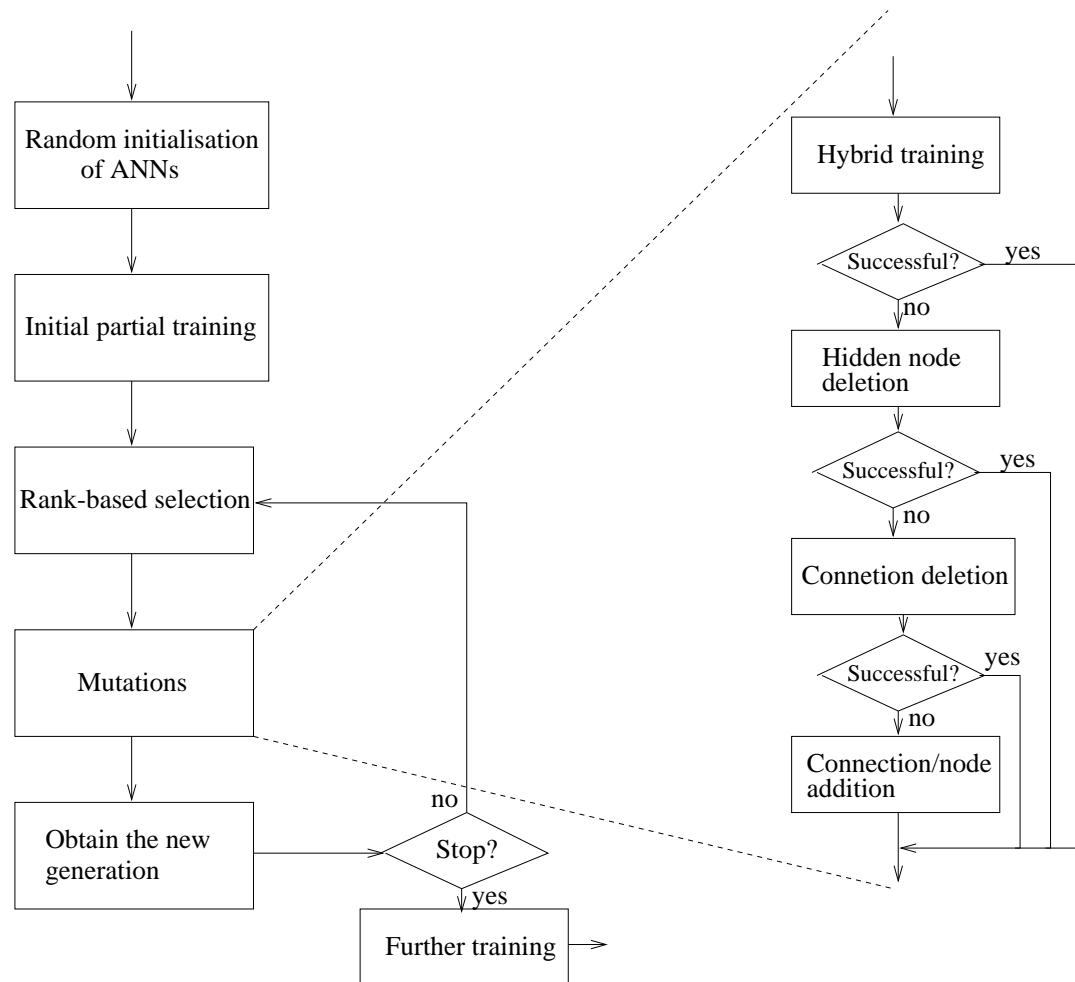
Problem 2: Behaviour Disruption by Genetic Operators

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.

An Alternative 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



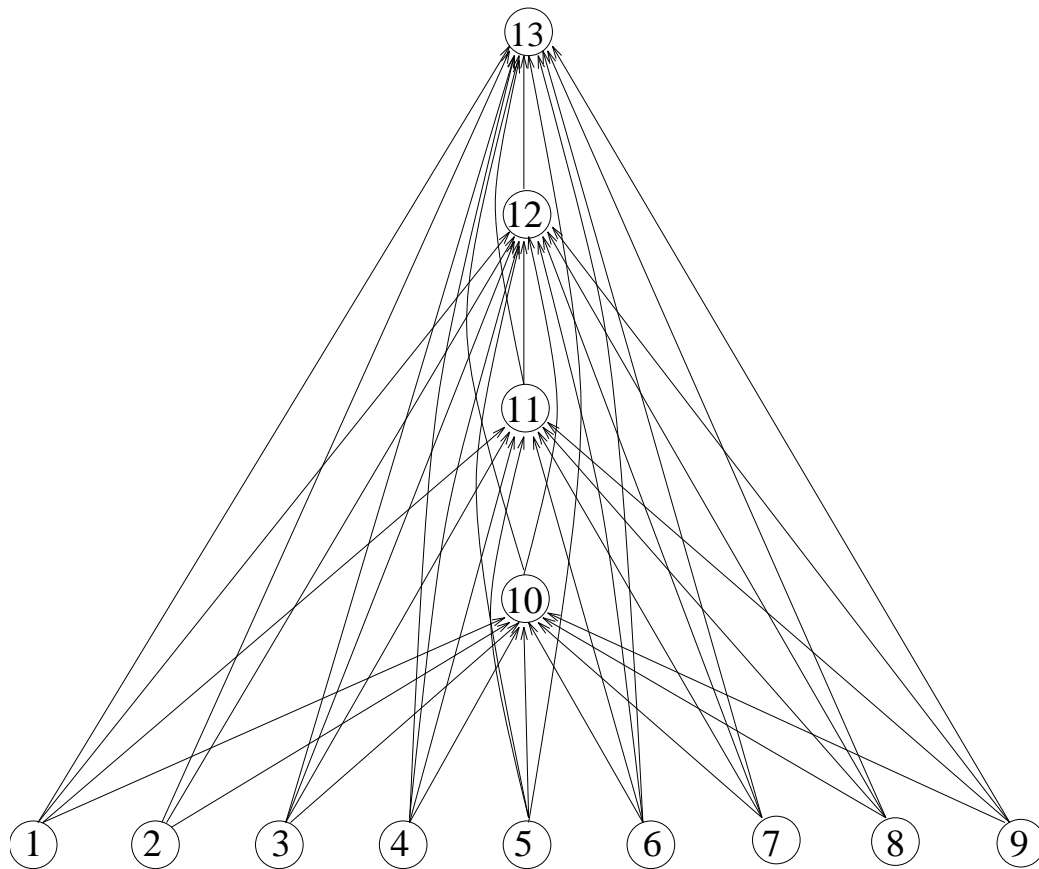
Mutation

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*.

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

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

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.

Experiments: Evolving a population of ANNs.

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

Linear Combination by RLS Algorithm

- The weights are computed by minimising the mean square error

$$E = \sum_{i=1}^M \left(d(i) - \sum_{j=1}^N w_j o_j(i) \right)^2 \quad (1)$$

where N is the population size, M is the number of training examples and $d(i)$ is the desired output for example i .

- We used the RLS (recursive least square) algorithm to minimise E .

Summary and Remarks

1. Chromosome representation has an important impact on the success of evolutionary learning and architectural design.
2. Transforming a problem from one space to another one may bring in certain benefits.
3. When the mapping between genotypes and phenotypes is not one-to-one, care must be taken in fitness evaluation.
4. Evolution and learning are two powerful forms of adaptation.

References (Downloadable)

1. X. Yao and Md. M. Islam, “Evolving artificial neural network ensembles,” *IEEE Computational Intelligence Magazine*, 3(1):31-42, February 2008.
2. X. Yao, “Evolving artificial neural networks,” *Proceedings of the IEEE*, **87**(9):1423-1447, September 1999.
3. X. Yao and Y. Liu, “A new evolutionary system for evolving artificial neural networks,” *IEEE Transactions on Neural Networks*, **8**(3):694-713, May 1997.