Genetic Algorithms and Genetic Programming

Lecture **6**: (13/10/09)

Hybrid algorithms

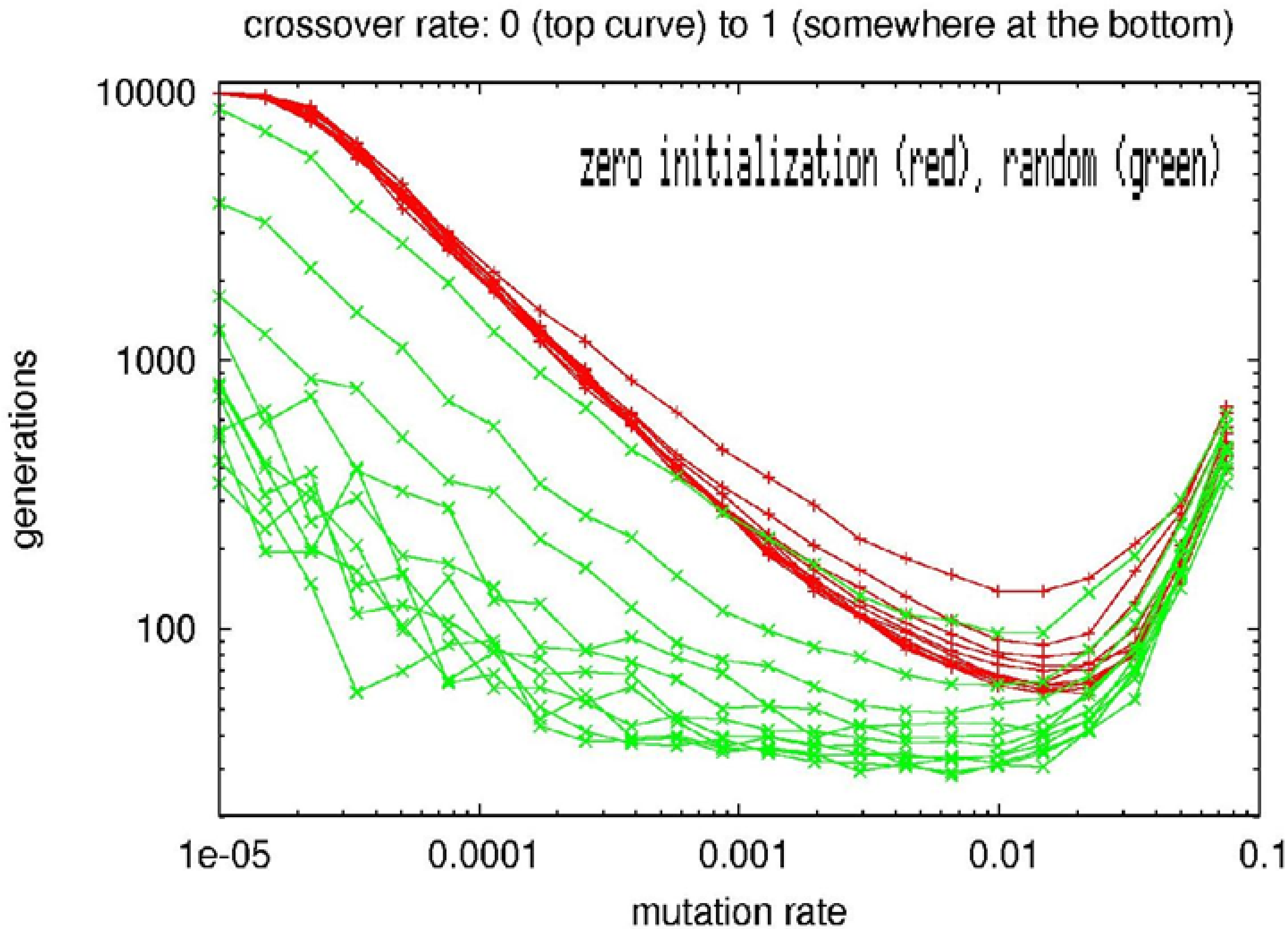
School of informatics

Michael Herrmann

michael.herrmann@ed.ac.uk, phone: 0131 6 517177, Informatics Forum 1.42

Overview

1. Introduction: History
2. The genetic code
3. The canonical genetic algorithm
4. Examples & Variants of GA
5. The schema theorem
6. Hybrid algorithms
7. Evolutionary robotics
8. GP



Number of generations required to discover the optimal solution
Strings of 20 characters c_i={0,1}, P=100, f(c) = Σ c_i (“all-ones” problem)
Initialization: a) c_i=0 with prob. ½ and c_i=1 otherwise or b) c=(0,…,0)

The Schema Theorem

$$E(m(H,t+1)) \geq m(H,t)\frac{\hat{u}(H,t)}{\bar{f}(t)}(1-p_c\frac{d(H)}{l-1})[(1-p_m)^{o(H)}]$$

Schema Theorem in words: short, low-order, above average schemata receive exponentially increasing trials in subsequent generations of a genetic algorithm.

Beyond the schema theorem:

- How do schemata arise?
  Constructive role of mutation and crossover
- Mean fitness changes if more fit individuals are around
  Other ways to change the fitness?
- Which genes belong to a good schema?
  The algorithm does not easily distinguish important genes from "hitchhikers"

When Do GAs Do Better Than Hill-climbing?

To act like an ideal GA and outperform hill-climbing (at least in this sort of landscape) need

- Independent samples: big enough population, slow enough selection, high enough mutation, so that no bit-positions are fixed at same value in every chromosome
- Keeping desired schemas: strong enough selection to keep desired schemas but slow enough selection to avoid hitch-hiking
- We want crossover to cross over good schemas quickly when they're found to make better chromosomes (but we don't want crossover to disrupt solutions)
- Large N/long string so speedup over RMHC is worth it.

Not possible to satisfy all constraints at once – tailor to your problem

Where Now?

- Schema theorem starts to give us an idea of how GAs work but is flawed $\rightarrow$ need better mathematical models of GA convergence . . .
- . . . but these better models don't make our GA go faster. Can we fix it empirically? Fix what, exactly?
- 1. Standard GA finds good areas, but lacks the **killer instinct** to find the globally best solution
- 2. Standard crossover often disrupts good solutions late in the run
- 3. Binary representations of non-binary problems often slow the GA down rather than allowing it to sample more freely. The "Hamming Cliff".

Aim is to shift balance from **exploration** at start to **exploitation** at end.

The Killer Instinct and Memetic Algorithms

- Hill-climbing local neighbourhood search is a fast single solution method which quickly gets stuck in local optima (cf. SAHC, NAHC)

- Genetic algorithms are a multi-solution technique which find good approximate solutions which are non-local optima

- Hence: try applying local search to each member of a population after crossover/mutation has been applied. We might find locally better solutions, and if near the end of run find the best/optimal solution

- GA + LS = Memetic Algorithm

Evolution theory

- Jean Baptist Lamarck: First truly cohesive theory of evolution [Inheritance of acquired characters] (around 1800)
- Charles Darwin: Natural selection
  (*On the Origin of Species*, 1859)
- Herbert Spencer: Survival of the fittest
  (*Principles of Biology*, 1864)
- 1866: Gregor Mendel: Rules of inheritance in pea plants
- 1905: “Genetics” (William Bateson)
- 1953: DNA structure (Crick and Watson)
- 1977: virus genome, 2003 Human genome (99%)

The Baldwin effect

- “A new factor in evolution” (James Baldwin, 1896)
- Selection for learning ability (rather than relying only on fixed abilities from the genes)
- Increased flexibility: Robustness to changes in the environment (i.e. changes of the fitness function)
- Learning has a cost:
  - If learning of the same tasks increases fitness over many generations then those individuals have a relatively higher fitness that produce (parts of) these results by their genetically fixed abilities
- Selective pressure may lead to a translation of learned abilities into genetic information!

Computational study: Hinton & Nowlan: How learning can guide evolution (1987)
(see M. Mitchell, Chapter 3)

Memetic Algorithms

- 1st generation: Hybrid algorithms
  - evolutionary algorithm + local refinement (development and learning)
- 2nd generation: Hyper-heuristic MA (Lamarckian)
  - includes evolution of the learning algorithm(s) by selection of memes
- 3rd generation: Co-evolution, self-generating MA
  - co-adaptation of the representation of memes including discovery of new memes

Hybrid GA: Evolving Neural Networks

- Reminder of neural networks
- Evolving weights
- Evolving network topology
- Grammars, robotics
- Evolving intelligent behaviours
- Example: evolving communication

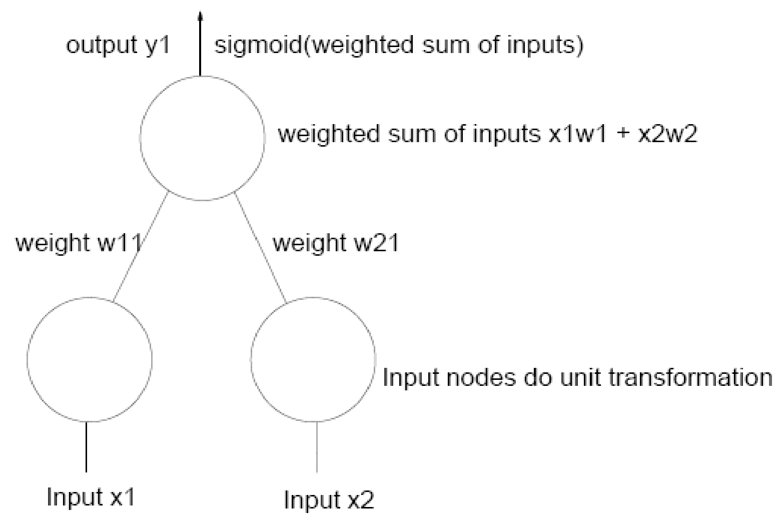
Neural Networks

- Inspired by working of neurons in the brain
- Universal function approximators
- Used widely in machine learning
- Empirical predictive modelling
- Often used for Classification
- Robotic controllers – input to network from sensors, output from network to motors

Basic Properties of Neural Networks

- Nodes and connections
- Weights attached to the connections
- Firing (output from node) depends on inputs to the node
- Nodes calculate the weighted sum of their inputs
- Activation threshold function
- Input/hidden/output layers. Each layer is fully connected to the next
- Feedforward vs. recurrent networks
- Training: back propagation

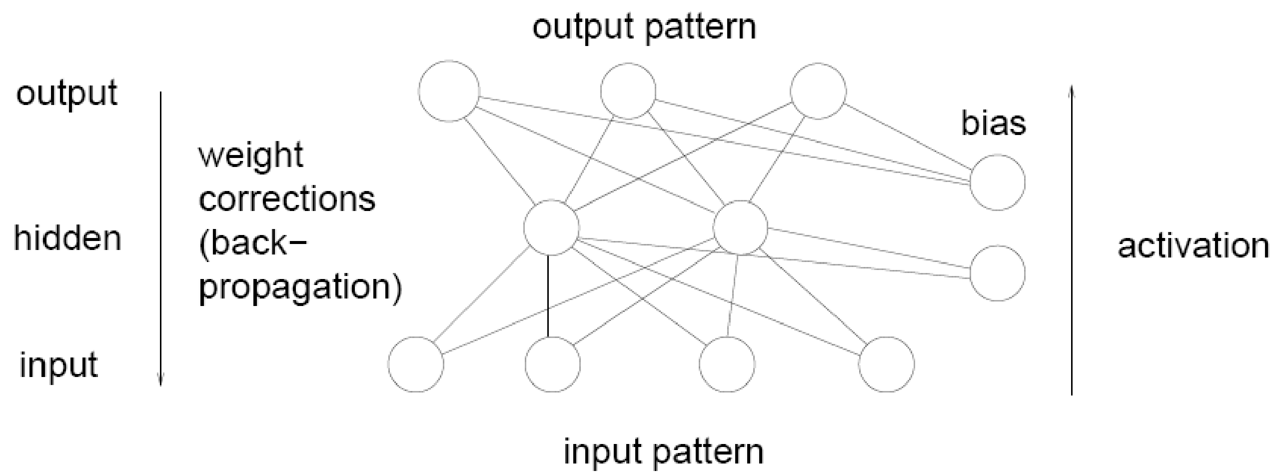
A Simple Feedforward Neural Network



Each node (apart from input nodes) takes the weighted sum of its inputs, and feeds this sum through a sigmoid function:

$$y_j = \frac{1}{1 + e^{-u_j}} \text{ where } u_j = \Sigma_i w_{ij} x_i$$

A Typical Feedforward Neural Network



Input, hidden and output nodes. Output from bias nodes is 1.
Learning procedure: use a training set of <input, output> pairs.
Present input, try to adjust weights to reduce the difference between the network's output and the desired output – backpropagation algorithm (Rumelhart et al. 1986)

– supervised learning procedure

Evolving Weights

- evolve the weights rather than train the network directly
- as an alternative to back-propagation

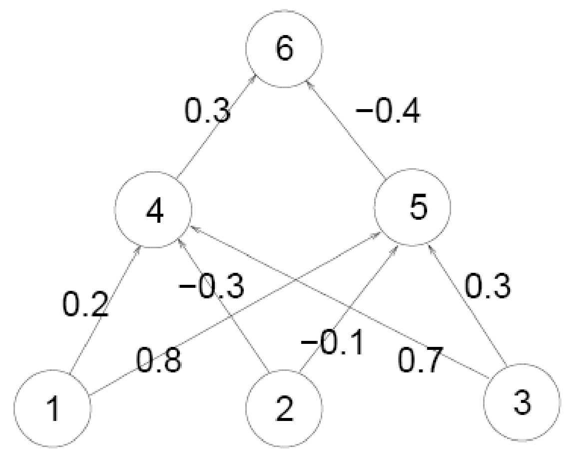
Montana and Davis (IJCAI 1989) looked at:

- underwater sonic recordings (features, preprocessed)
- treated as a classification problem (whales, enemy subs)
- network topology

4 input nodes	
7 nodes in hidden layer 1	fully connected
10 nodes in hidden layer 2	18 extra thresholding connections (biases)
1 output node	total weights 126

- GA chromosome: a list of 126 real-valued weights

Represent the Weights on a Chromosome



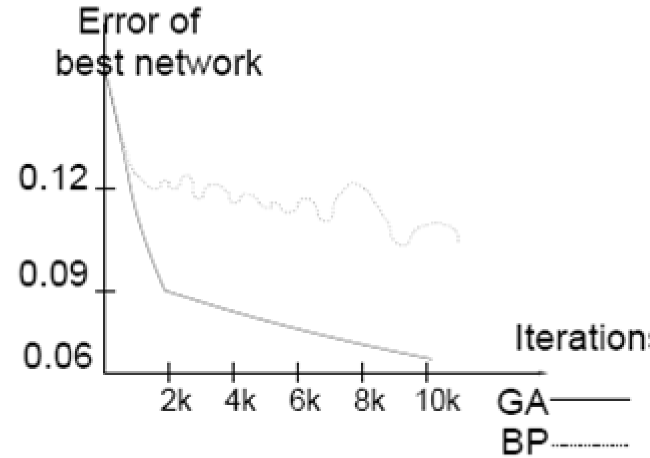
Chromosome: (0.3, -0.4, 0.2, -0.3, 0.7, 0.8, -0.1, -0.3)

Building blocks: all incoming weights to a given unit seems plausible.

Mutation: for **each** link coming in to the chosen node, add a (different) random value between +1.0 and -1.0

Crossover: for each non-input node, choose **all** the weights from Parent 1 or **all** the weights from Parent 2. (Montana-Davis crossover)

Results of Weight Evolution



Reward function: how well the actual network output matches the training output over the training set

- GA were better than BP for some tasks
- 'unsupervised' learning – we're not changing a single network to be more likely to produce the right output, we're evaluating a network then throwing it away and producing the next generation
- good if sparse reinforcement available, e.g. if network controls a robot moving

  in unfamiliar environments – we may only need it to work in some parts of the input/output space, i.e. those actually experienced. So we evaluate its fitness as a controller in just those bits of the environment where we need to run it
- backpropagation doesn't work well if subsequent inputs are correlated, so GA may be better

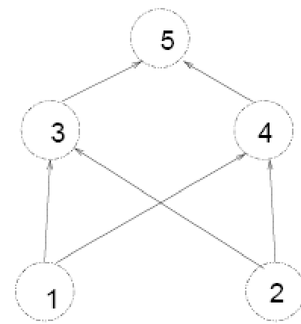
Evolving Topology 1

– choosing a network topology is hard

– can it be done automatically?

Miller, Todd and Hegde (1989):

from unit:		1	2	3	4	5
to unit:	1	0	0	0	0	0
	2	0	0	0	0	0
	3	1	?	?	?	?
	4	1	?	?	?	?
	5	0	?	?	?	?



0 if not connected
1 if connected = learnable
? = you complete the table

Chromosome: 00000 00000 ... (complete the rest...)

Mutation: bit flipping

Crossover: exchange whole rows

Limit to **feedforward networks**: any links to input units or feedback connections are ignored.

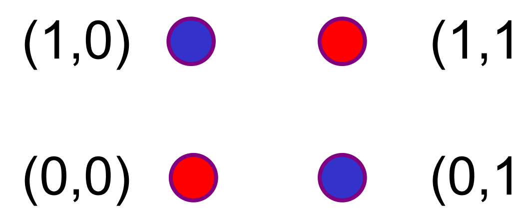
Evolving Topology 2

Tasks tried by Miller et al.:

(a) XOR (exclusive - OR)

(b) four quadrant:

$$(x,y) \rightarrow 0.0 \ \text{ if } \ x, y \simeq 0.0 \ \text{ or } \ x, y \simeq 1.0$$
$$(x,y) \rightarrow 1.0 \ \text{ otherwise}$$



(c) pattern copying, with units in the hidden layer $<$ number of input units

Learning: back-propagation

Results: GA can easily find network topologies for these problems.

But are the problems too easy?

See Stanley and Miikkulainen (2002) for a more sophisticated approach (NEAT)

NEAT Neuro-Evolution through Augmenting Topologies

• Evolve by changing the connection weights, turning links on and off, and also by adding nodes and links (in the mutation stage)

Start off simple, become more complex – as complex as needed – complexification

If a node is added it is added in the middle of an existing link

• Crossover: need to match up parts of the network coding for similar traits.

Competing conventions: permuting a network doesn't change the outputs or the function computed by the network

So: give each gene an *innovation number* – the next unused integer when it is made. So can match up parts of networks inheriting this gene in future generations.

NEAT Neuro-Evolution through Augmenting Topologies

Inherit matching genes from each parent with equal probability. Inherit non-matching genes from fittest parent.

- Can also split into species based on difference between chromosomes (based on number of matching genes and other metrics). Preserves new topology for a while so that it has a chance to optimise its structure only in competition with similar members.
- Works well on pole-balancing. Also applied to game of GO.

See Kenneth O. Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. Evolutionary Computation 10, 99–127 (2002)

Grammars and Robotics

- Grammatical encoding of the linkage matrix (here for XOR)

$$
S \rightarrow \begin{matrix} A & B \\ C & D \end{matrix} \rightarrow \begin{matrix} c & p & a & a \\ a & c & a & e \\ a & a & a & a \\ a & a & a & b \end{matrix} \rightarrow \begin{matrix} 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{matrix}
$$

(S A B C D | A c p a c | B a a a e . . .)

Generate linkage matrix from the grammar. If at the end of rewriting there are still non-terminal nodes, that node is "dead" – not connected.

• Develop chromosome (genotype) into network (phenotype) and train for fixed no. of training episodes.

• Fitness = error at end of training

Problems with direct encoding

Fixed connections: as size of matrix grows, chromosome size grows

Can't encode repeated patterns, esp. with internal structure

Takes a long time to generate high-performing networks

Advantages of grammatical encoding

Can represent large connectivity matrices in compact form

Shorter encoding, faster search

Variable topologies including recurrent connections

Better on encoder/decoder problem than direct encoding

Evolving Neural Network Behaviours

- Previous examples rely on *training data*
- What if we haven't got any?
- Example: a neural network which controls a mobile agent which is trying to achieve some goals in a dynamic environment.
- No good example of behaviour is available; or we wish to try a range of possible behaviours to see which is best.
- A fitness function is available based on goal achievement.

Evolving Neural Network Behaviours

General approach:

- Decide on how to represent inputs to and outputs from the neural network.
- Decide on a neural network architecture: might need to try a range of possibilities.
- Decide on a simulation which tests the NN's behaviour.
- Decide on a fitness function which tests how well the NN did in the simulation.
- All the usual GA stuff: chromosome representation, crossover, mutation, population size, etc.

Example: Evolving Communication

This is an example from Artificial Life: the study of computer generated "life" forms. (Matthew Quinn, University of Sussex)

- Khepera robots controlled by evolved neural networks
- Group task: robots move together as far as possible – like dancing
- 8 sensor nodes, 4 motor nodes, hidden nodes
- Evolved thresholds, weights, decay parameters, size, connectivity of network
- **Co-evolution**: select two robots from population, rate them for fitness *as a pair*
- Initial result: leaders and followers emerge
- Only get a working pair 50% of the time, BUT....

Example: Evolving Communication

- After a while a new single species emerges

- This behaviour uses communication based on simple movement:
  - both agents (A and B) rotate anti-clockwise
  - one agent (B) becomes aligned first and moves towards the other agent
  - agent B moves backward and forward while staying close to A
  - when A becomes aligned, it becomes the leader: it reverses its direction and is followed by B

- Very similar to movement communication used in social insects (e.g. dancing in honey bees)