Non-Symbolic AI – Summer 2006

EASy

Lecturer: Inman Harvey PEV2 rm 5C12 x8431

inmanh@susx.ac.uk

www.informatics.susx.ac.uk/users/inmanh/non-symb

Lectures:

- Tue 11:00 Thu 16:00 Fri 9:00 in ARUN-401

Seminars – split into groups – start in Week 2:

- Thu 09:00 in PEV1-1A3
- Fri 14:00 in PEV1-1A1

Objectives

EASy

- Familiarity with a broad range of non-symbolic AI
- NSAI for cognition – in robots or software
- Neural Nets (NNs) for cognition – eg robotics
- NNs for data-mining and applications
- Genetic Algorithms (GAs) for design
- GAs for data-mining and applications
- Ability to program GAs and NNs for these purposes

Prerequisites

EASy

It is assumed that you have some experience of background AI concepts to the course, eg from Further AI.

A lot of topics will be similar to Further AI, covered differently.

It is assumed that you can write programs in an appropriate language.

It is assumed that you can pursue topics through further reading, discussing with colleagues and asking questions in seminars!

Seminars

EASy

Seminars will start in week 2, and different seminars may be taken by different people.

Topic for week 2 will be based on a paper by

R. Pfeifer (1996): Building 'Fungus Eaters': Design Principles of Autonomous Agents. SAB96.

This paper will be made available this week, and you are expected to read it before hand, so that any of you can be called on to present the ideas in the paper.

Seminar lists

EASy

Your groupings into the different seminar slots will be announced shortly – and like everything else, will be kept up-to-date on

www.informatics.susx.ac.uk/users/inmanh/non-symb

Week 2: Seminar based on Reading

Week 3: GA exercise ***

Week 4: Backprop ANN exercise ***

Week 5: Seminar based on Reading

Seminars -> Assessed Coursework

EASy

Seminars in weeks 3 and 4 are GA and ANN exercises

You are expected to make a proper attempt on these before, and bring to the seminars, for feedback.

These exercises do not count towards the assessed coursework ***BUT*** by the time you have done them and put them together, you will find that most of the assessed coursework exercise is pretty much done already ***!!!!!!***

Reading List

EASy

For Robotics and Autonomous Systems:

Understanding Intelligence Pfeiffer & Scheier, MIT Press 1999

For Genetic Algorithms

An Introduction to Genetic Algorithms Mitchell, MIT Pr 1996

For Neural Networks

Neural Computing Beale & Jackson, Adam Hilger 1990

Other Reading

EASy

Designing Autonomous Agents, P. Maes (MIT)

Artificial Life, C. Langton (MIT)

An Intro to Neural Networks, J. Anderson (MIT)

Neural Networks for Pattern Recognition, CW Bishop (OUP)

Genetic Algorithms in Search … D. Goldberg (Addison-Wesley)

From Animals to Animats (Series of conference proceedings for SAB conferences).

Lecture Notes

EASy

… can be got as a complete term pack from Celia in COGS Library

… and will also be posted on website

www.informatics.susx.ac.uk/users/inmanh/non-symb

These are **not**, however, a substitute for attending the lectures and seminars!

Assessment

EASy

The course is assessed by 50% coursework and 50% unseen exam.

The **exam** will be some time in June – look out for announcements. You should answer 2 out of the 3 available questions within the 1.5 hours.

The **coursework** will be a programming exercise, with a short report (maximum 2000 words), that will be set in week 2, to be handed in by Thurs May 25 (Wk 6).

There are big penalties for handing in work late (10% up to 24 hrs late, **then 100% !!!**) so you should plan to complete in good time.

Outline of lectures

EASy

Intro + 2 lectures on Genetic Algorithms

3 lectures on Alife and Robotics

3 lectures on Neural Networks (with some data-mining and more GAs)

2 lectures on this and that (coevolution, communication…)

Extra lectures at end – I will ask for suggestions as to either covering a new topic that you want, or returning and covering in more depth something previous – you will decide.

Type of Lectures

EASy

Some lectures will cover tricky stuff, at a rather abstract and hand-wavy level. For these topics, you will be expected to pick up the general flavour without necessarily getting to grips with the detail (… unless of course you want to).

Other lectures will be covering topics such as GAs and Neural Networks at a low and simple level – for these topics you will be **expected** to be able to program some versions of these by the end of the course.

What is Non-Symbolic AI?

1. What is AI ?
2. What is Symbolic AI ?
3. What is Non-Symbolic AI ?

The difference between 2 and 3 will be indicated by a rapid history of 2000 years of AI !

What is AI ?

I am going to distinguish 2 (connected) parts to AI :-

1. Building hardware or software (robots or programs) that replicates (some aspects) of intelligent, adaptive behaviour as seen in humans or animals – e.g. trying to pass (some version of) the Turing test. Cognitive Science
2. Building tools to help humans tackle specific jobs in ways that need intelligence – e.g. data-mining, useful software tools, robots.

Crudely, these are **Science** and **Engineering.**

AI and Alife

AI has tended to concentrate on **logic**, on **calculation**, on **formal systems** as the kind of intelligence to emulate in machines.

But recently – particularly with the new field of **Artificial Life** (Alife) – people have widened their ideas of what counts as 'intelligence'. The ability of a bird to navigate between N. Europe and S. Africa is amazing, displays some kind of adaptive intelligence – but does it use logic?

Early Artificial Life

A whirlwind tour through 2 millennia.

*** Chapter 1 of Artificial Life, Chris Langton (ed), Addison Wesley 1989. Proc of First workshop on Artificial Life.

Automata
Started with the Ancient Greeks. 1st century AD, Hero of Alexandria described working models of animals and humans, using
hydraulics and pneumatics.

Middle Ages

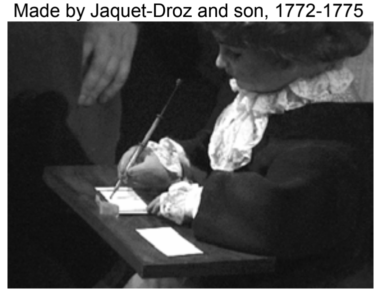
From around 14th Century AD, development of clocks allowed more sophisticated automata.

Early Alife quote:
"For seeing life is but a motion of Limbs, the beginning whereof is in the principal part within; why may we not say that all *Automata* (Engines that move themselves by springs and wheeles as doth a watch) have an **artificiall life**?"

Thomas Hobbes in *Leviathan* (1651)

18^{th} C Automata

Made by Jaquet-Droz and son, 1772-1775

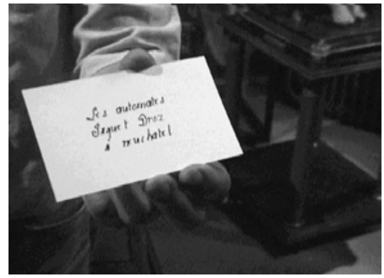


18th C Automata (2)

EASy

18th C Automata (3)

EASy



18th C Automata (4)

EASy

18th C Automata (5)

EASy

Jump to 20 C

EASy

2nd World War – **Cybernetics** "the study of control and communication in the animal and machine" N Wiener. Aiming of anti-aircraft fire -- notion of **Feedback**

A lot of important early work in Cybernetics in 1940/50s that got rather forgotten in the rise of **Computing**.

Well worth searching for this early Cybernetics work -- I consider **Design for a Brain**, by **W Ross Ashby**, Wiley & Sons 1952, enormously important.

And Computing

EASy

Then came computing ... ... the classical AI approach ... disembodied abstract reasoning.

Computing has been enormously successful for abstract problem solving, but led to this insidious popular view that humans and animals think and behave like problem-solving computers.

Embodied behaviour before abstract rationality

EASy

From several directions, particularly in the last decade, has come the realisation that humans are the product of 4 billion years of evolution, and only the last tiny fraction of this period has involved language and reasoning.

If we dont understand the capacities of simple organisms, how can we hope to understand human capacities?

Cf. Rod Brooks, robot subsumption architecture.
This is **one motive** for doing A-life. **(RB talk 14 May)**

OK, so what is Artificial Life?

EASy

"Artificial Life is the study of man-made systems that exhibit behaviors characteristic of natural living systems. It complements the traditional biological sciences concerned with the *analysis* of living organisms by attempting to *synthesize* life-like behaviors within computers and other artificial media. By extending the
empirical foundation upon which biology is based *beyond* the carbon-chain life that has evolved on Earth, Artificial Life can contribute to theoretical biology by locating *life-as-we-know-it* within the larger picture of *life-as-it-could-be."*

Chris Langton (in Proc. of first Alife conference)

Alife as conscious echo of AI

EASy

Note 2 meanings of 'Artificial':
(1) = fake (eg artificial snow)
(2) = made by artifice, an artefact, but not fake
(eg artificial light)

Two positions you will come across:
Weak Alife: computer programs as useful simulations of real life
Strong Alife: ditto as **actually living**

Non-Symbolic AI (1)

EASy

So, one aspect of Non-Symbolic AI (maybe the ‘Science’ part) is an extension of ideas of Intelligence to include all sorts of adaptive behaviour, not just the ‘rational’ part of human behaviour.

After all, in the 4 billion year history of our species, rationality only ‘turned up’ fairly recently, and even now we mostly get by without using logic!

This part of Non-Symbolic AI is demonstrated in Alife, in situated embodied robotics, etc.

Non-Symbolic AI (2)

EASy

But there is a 2nd aspect to n-sAI (maybe the Engineering part).

This comes from recognising that symbolic AI approaches to eg pattern recognition are useless in comparison to the ability of a migrating bird (that does not use symbols or logic)

… that the most complex bit of machinery humans have designed is trivial (in performance, in efficiency, in robustness) compared to even the simplest natural organism.

So let’s try and understand and borrow some of Nature’s tricks.

Nature’s tricks (1)

EASy

In particular, for jobs such as pattern-recognition (is that where I turn left to go home? Is that a crack in the wing? Is that a tumour on this X-ray? Is that a sign that the stock-market is about to crash?), maybe we can get some ideas from Neural Networks.

Artificial Neural Networks (ANNs) come in all sorts of varieties, and one class (which may or may not be similar to natural NNs) is potentially useful for pattern-recognition tasks.

Feedforward, multilayer perceptrons, backprop etc

Nature's tricks (2)

EASy

Another class of ANNs borrows from the role of real NNs in **control** – how sensors and motors are coordinated in action and perception.

Dynamic Recurrent NNs

Evolutionary Robotics

Brooks' subsumption architecture, though not usually described as an ANN, actually has some similarities with this sort of approach.

Nature's tricks (3)

EASy

Evolution is Nature's trick for designing complex interacting creatures – hence Evolutionary Robotics borrows directly from this.

More generally, Genetic Algorithms (GAs) are efficient search methods for finding design solutions to intricate problems (how can I organise the lecture timetable without clashes? How can I design an ANN for a robot brain? How can I find a simple formula to predict the weather, the horse that will win the 2:30 race at Newmarket?)

Next lecture will be on GAs.

Non-Symbolic AI

EASy

More generally, (and with prejudice!):

- Symbolic AI has its place, is crucially6 important for many machine learning techniques -- but has its limits as a model for how humans and animals actually behave
- Non-Symbolic AI, Alife, Evolutionary and Adaptive Systems, -- this is where currently much of the interesting new ideas and research is
- This is where there is currently a large demand for people with experience and skill.

Non-Symbolic AI Lecture 2

EASy

Evolution and Genetic Algorithms

Much of Non-Symbolic AI is borrowing from Nature's tricks. Perhaps **the** most important is the role of Darwinian Evolution, in designing all natural creatures around us – including you yourself!

Genetic Algorithms (GAs)

Biological Evolution

EASy

Read (strongly recommended, readable and fresh) the original C. Darwin 'On the Origin of Species'

Also John Maynard Smith 'The Theory of Evolution'

Richard Dawkins 'The Selfish Gene' etc.

M Ridley "Evolution" – (textbook)

Evolution

EASy

The context of evolution is a **population** (of organisms, objects, agents ...) that survive for a limited time (usually) and then die. Some produce **offspring** for succeeding generations, the **'fitter'** ones tend to produce more.

Over many generations, the make-up of the population changes. Without the need for any individual to change, successive generations, the 'species' changes, in some sense (usually) adapts to the conditions.

3 Requirements

EASy

✓**HEREDITY** - offspring are (roughly) identical to their parents

✓**VARIABILITY** - except not exactly the same, some significant variation

✓**SELECTION** - the 'fitter' ones are likely to have more offspring

Selection

EASy

Variability is usually **random** and **undirected**

Selection is usually **unrandom** and **directed**

In natural evolution the 'direction' of selection does not imply a **conscious Director** -- cf Blind Watchmaker.

In artificial evolution often **you** are the director.

Neo-Darwinism

EASy

Darwin invented the theory of evolution without any modern notion of genetics - that waited until Mendel's contributions were recognised.

neo-Darwinian theory = Darwin + Mendel + some maths
(eg Fisher, Haldane, Sewall Wright)

(Artificial) Genetics

In Genetic Algorithm (GA) terminology, the **genotype** is the full set of genes that any individual in the population has.

The **phenotype** is the individual potential solution to the problem, that the genotype **'encodes'.**

So if you are evolving with a GA the control structure, the 'nervous system' of a robot, then the genotype could be a string of 0s and 1s   001010010011100101001 and the phenotype would be the actual architecture of the control system which this genotype encoded.

An example

It is up to **you** to design an appropriate encoding system.

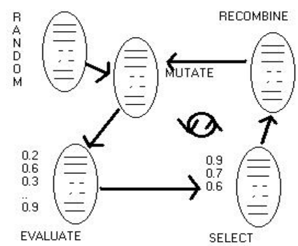
Eg. Evolving paper gliders

Fold TL to BR towards you Fold horiz middle away Fold vertical middle towards	Fold TR to BL towards you Fold horiz middle away Fold vertical middle away

Evolving paper gliders

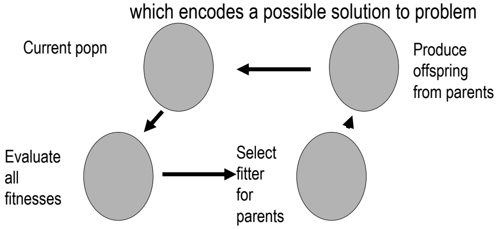
1. Generate 20 random sequences of folding instructions
2. Fold each piece of paper according to instructions written on them
3. Throw them all out of the window
4. Pick up the ones that went furthest, look at the instrns
5. Produce 20 new pieces of paper, writing on each bits of sequences from parent pieces of paper
6. Repeat from (2) on.

Basic Genetic Algorithm



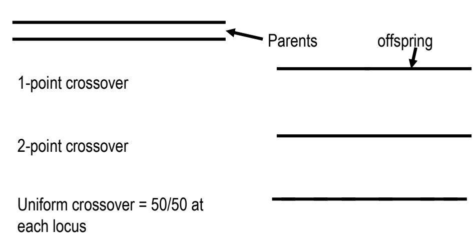
Basic GA -- continued

——— represents a genotype, string of characters, which encodes a possible solution to problem



Recombination

Typically 2 parents recombine to produce an offspring



Mutation

EASy

After an offspring has been produced from two parents (if sexual GA) or from one parent (if asexual GA)

Mutate at randomly chosen loci with some probability

Locus = a position on the genotype

A trivial example

EASy

Max-Ones – you have to find the string of 10 bits that matches as closely as possible this string: 1111111111

… and yes, clearly the answer will be 1111111111, but pretend that you don't know this. A fitness function:-

```
int evaluate(int *g) {
        int i,r=0;
        for (i=0;i<10;i++) r += (g(i) == 1);
        return(r);
}
```


Program structure

EASy

Initialise a population of (eg) 30 strings of length 10

```
int popn[30][10];
void initialise_popn() {
        int i,j;
        for (i=0;i<30;i++)
                for (j=0;j<10;j++)
                        popn[i][j]= flip_a_bit();
}
```


Main Program Loop

EASy

```
For n times round generation loop
        evaluate all the population (of 30)
        select preferentially the fitter ones as parents
        for 30 times round repro loop
                pick 2 from parental pool
                recombine to make 1 offspring
                mutate the offspring
        end repro loop
        throw away parental generation and replace with offspring
End generation loop
```


Variant GA methods

EASy

We have already mentioned different recombination (crossover) methods - 1-pt, 2-pt, uniform.

You can have a GA with no recombination -- asexual with mutation only.

Mutation rates can be varied, with effects on how well the GA works.

Population size -- big or small? (Often 30 - 100)

Selection Methods

EASy

Eg. **Truncation Selection**.
All parents come from top-scoring 50% (or 20% or ..)

A different common method: **Fitness-proportionate**
If fitnesses of (an example small) population are
2 and 5 and 3 and 7 and 4 total 21
then to generate each offspring you select mum with
2/21 5/21 3/21 7/21 4/21 probability
and likewise to select dad. Repeat for each offspring.

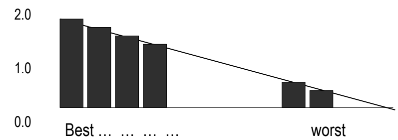
Different Selection Methods

Problems with fitness-proportionate:
✓**How about** negative scores ?
✓**How about** if early on all scores are zero (or near-zero) bar one slightly bigger -- then it will 'unfairly' dominate the parenting of next generation?
✓**How about** if later on in GA all the scores vary slightly about some average (eg 1020, 1010, 1025, 1017 ...) then there will be very little selection pressure to improve through these small differences?

You will see in literature reference to scaling (eg sigma-scaling) to get around these problems.

Rank Selection

With linear rank selection you line up all in the population according to rank, and give them probabilities of being selected-as-parent in proportion:



More Rank Selection

Note with linear rank selection you ignore the absolute differences in scores, focus purely on ranking.

The 'line' in linear ranking need not slope from 2.0 to 0.0, it could eg slope from 1.5 to 0.5.
You could have non-linear ranking. But the most common way (I recommend unless you have good reasons otherwise) is linear slope from 2.0 to 0.0 as shown.

This means that the **best** can expect to have **twice** as many offspring as the **average.** Even below-average have a sporting chance of being parents.

Elitism

Many people swear by elitism (...I don't!)

Elitism is the GA strategy whereby **as well as** producing the next generation through whichever selection, recombination, mutation methods you wish, you also **force** the direct unmutated copy of best-of-last generation into this generation -- 'never lose the best'.

Genotype Encoding

Often genotypes in GA problems have **discrete** characters from a **finite** alphabet at each locus.

Eg. 0s and 1s for a binary genotype 010011001110
-- a bit like real DNA which has 4 characters GCAT

These often make sense with simple encodings of strategies, or connectivity matrices, or …

Coding for Real Numbers

But sometimes you want to solve a problem with **real** numbers -- where a solution may include 3.14159

Obvious solution 1: binary encoding in a suitable number of bits. For 8-bit accuracy, specify max and min possible values of the variable to be coded.
Divide this range by 256 points.

Then genes 00000000 to 11111111 can be decoded as 8-bit numbers, interpolated into this range.

Coding for Many Real numbers

EASy

For eg 10 such real-valued variables, stick 10 such genes together into a genotype 80 bits long.
You may only need 4-bit or 6-bit accuracy, or whatever is appropriate to your problem.

A problem with binary encoding is that of '**Hamming cliffs**'

An 8-bit binary gene 01111111 encodes the next value to 10000000 -- yet despite being close in real values, these genes lie 8 mutations apart (a Hamming distance of 8 bits)

Gray Coding

EASy

This is a 1-1 mapping which means that any 2 adjoining numbers are encoded by genes only 1 mutation apart (tho note reverse is not true!) -- no Hamming Cliffs

Rule of thumb to translate binary to Gray:
Start from left, copy the first bit, thereafter when digit changes write 1 otherwise write 0.

Example with 3 bit numbers :--

Bin	Actual	Gray
000	0	000
001	1	001
010	2	011
011	3	010
100	4	110
101	5	111
110	6	101
111	7	100

Other Evolutionary Algorithms

EASy

Note that GAs are just one type of evolutionary algorithm, and possibly not the best for particular purposes, including for encoding real numbers.

GAs were invented by John Holland around 1960s
Others you will come across include:

EP Evolutionary Programming
originally Fogel Owens and Walsh,
now David Fogel = Fogel Jr.

And more …

EASy

ES Evolution Strategies invented in Germany
Rechenberg, Hans-Paul Schwefel
Especially for optimisation, real numbers

GP Genetic Programming
Developed by John Koza
(earlier version by N Cramer).

Evolving programs, usually Lisp-like, wide publicity.

Which is best ?

EASy

Is there a universal algorithm ideal for all problems
-- **NO** !!

(cf 'No Free Lunch Theorem, Wolpert and MacReady)

Are some algorithms suitable for some problems
-- **PROBABLY YES**.

Is this a bit of a Black Art, aided by gossip as to what has worked well for other people -- **YES!**

Recommendation …

EASy

For Design Problems, encoding discrete symbols rather than reals, **my own initial heuristic** is:

GA (usually steady state rather than generational…)
selection: linear rank based, slope 2.0 to 0.0
sexual, uniform recombination
mutation rate very approx 1 mutation per genotype (**NB next slide)**
no elitism
population size 30 - 100
But others will disagree...

Mutation rates

EASy

Last slide I recommended mutation rates of around 1 per genotype per generation. I should stress this is when you are using **binary genotypes**, and assumes standard selection pressures and no redundancy – should be adjusted if there is non-standard selection and/or much redundancy.

If you are using real-valued genotypes, then probably mutation can alter **all** the loci 'a little bit'. Think in terms of a vector in n-dimensional space, mutation shifts it a bit.

Sources of Information

EASy

David Goldberg 1989 "Genetic Algorithms in Search, Optimization and Machine Learning" Addison Wesley

Melanie Mitchell and Stephanie Forrest "Genetic Algorithms and Artificial Life".
Artificial Life v1 no3 pp 267-289, 1994.

Melanie Mitchell "An Intro to GAs" MIT Press 1998

Z Michalewicz "GAs + Data Structures = Evolution Programs" Springer Verlag 1996

More …

EASy

plus many many more sources eg…

news group comp.ai.genetic

Be aware that there are many different opinions – and a lot of ill-informed nonsense.

Make sure that you distinguish GAs from EP ES GP.

Non-Symbolic AI lecture 3

EASy

More on Evolutionary Algorithms
Last lecture discussed encoding **real** numbers as **bits** on a genotype (either binary encoding or Gray coding)

Sometimes people choose to have real numbers directly represented on the genotype -- which might be:
2.034 -30.678 0.005 102.567 ... ... -89.432

Recombination will work in the same way as with normal discretely encoded genotypes, but **mutations** will be handled **differently.**

Mutating Real numbers

EASy

Various possibilities for **mutating** real numbers

One possibility is to change any mutated locus to a randomly chosen real number within the appropriate range -- but this is very **disruptive.**

So more usually a form of '**creep mutation**' is used:
eg. add a random number in range [-0.1 +0.1]
or add a random number drawn from a **Gaussian** distribution with mean zero, and appropriate range.

Evolution Strategies

EASy

If the problem you are tackling has all the parameters naturally expressed as real numbers, then maybe you should investigate Evolution Strategies
(see previous lecture)

These work primarily with a version of 'creep mutation', and this evolutionary paradigm has developed sophisticated strategies for **modifying** the amounts of 'creep' in different dimensions as evolution proceeds.

Different Search Algorithms

EASy

Or indeed you could look at **Simulated Annealing**
-- a non-evolutionary technique which nevertheless has some similarities.

These are all techniques for Search within a many-dimensional, **real-valued** Search Space.

Genetic Algorithms may be more appropriate for Search within high-dimensional **Discrete** Search Spaces.

Many *design* problems are such – but many are not.

Why Should GAs work ?

EASy

John Holland (1975) 'Adaptation in Natural and Artificial Systems' -- and most of the textbooks -- explain this with the **Schema Theorem**, and ideas of **building blocks.**

Roughly speaking, building blocks are segments of the genotype which encode for functional components of the 'phenotype', or potential solution to the problem.

These building blocks can, in principle, be evaluated independently of all the rest, as varying between 'good' and 'bad'.

Cartoon Version

EASy

Cartoon version of genotypes:

*** long legs *****************short arms**********

short legs************** long arms **********
^

Recombination (when crossover happens to land appropriately) allows different parents like these **in one generation** to produce a child with long legs **and** long arms

Schemata

EASy

Schemata (plural of schema) are a formalisation of this idea of a building block.

Consider binary genotypes of length 16. Let # be a **'wild-card'** or **'dont-care'** character.

Then #####00#010#####

is a schema of **order** 5 (5 specified alleles) and of **defining length** 6 (length of segment which includes specified alleles).

'Processing Schemata'

EASy

Considering this schema
#####00#010#####
then
0000000001010000
is just **one** of many genotypes corresponding to this schema -- and actually this genotype also corresponds simultaneously to **many** other schemata.

Implicitly, the GA **'evaluates'** and **'processes'** loads of schemata **in parallel,** every generation.

The Schema Theorem claims …

EASy

... that **schemata** of **short defining lengths** (coding for building blocks such as 'cartoon legs') will,
✓**IF** they are of above-average fitness, (..that is, evaluated whatever the other loci outside the schema are)
✓get **exponentially** increasing numbers of trials in successive generations.

Ie, **despite** recombination and mutation being **'disruptive'** (tho not too disruptive of short schemata) 'good building blocks' will multiply and **take over** --- and '**mix and match**' with other 'good building blocks'.

Implications of the Schema Theorem ??

EASy

The Schema Theorem is **formally** proved subject to certain conditions.

This Theorem is widely **interpreted** as implying that **RECOMBINATION** is the **'powerhouse'** of GAs,

-- whereas mutation is just a 'background operator' (whose only role is to add variety in loci where, throughout the whole population, no variety is left).

Doubts about the Schema Theorem

EASy

The Schema Theorem is formally **correct.**

But nowadays many people (including myself) believe it has been **misinterpreted.**

The 'subject to certain conditions' bit means that this exponential increase is only guaranteed over 1 generation

-- thereafter the conditions **change!**

Recombination versus Mutation ?

EASy

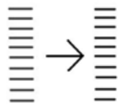
So be aware that despite this common view in the textbooks, some people think that in some sense **MUTATION** is the **powerhouse** of GAs, with recombination as a background (tho often useful) genetic operator.

"The Schema Theorem is true, but not very significant"

Nevertheless, the common view of the importance of recombination lies behind the exclusive emphasis (often without any mutation) on recombination in GP = Genetic Programming.

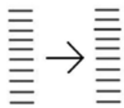
Some more GA wrinkles

You need not have a **generational** GA (where the whole population is swept aside every generation, and replaced by a fresh lot of offspring).



You can have a **STEADY STATE** GA.

Here just **ONE** member of the population is replaced at each time step, by the offspring of some others.



Steady State GA

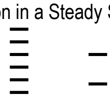
Eg with a popn of 100:
✓Choose a mum by some selection mechanism biased towards the fitter.
✓Choose a dad by same method.
✓Generate a child by recombination + mutation
✓Add the child to the population
✓Keep the numbers down to 100 by choosing someone else to die (eg at random, or biased towards the less fit)

Roughly speaking, 100 times round this loop is equivalent to one generation of a generational GA

Tournament Selection

Here is a very simple way to implement the equivalent of linear rank selection in a Steady State GA

Pick 2 at random



Fittest of tournament is mum – choose dad the same way

Generate offspring from mum and dad – the new offspring replaces someone chosen at random.
Note: everyone else remains, including mum and dad.
Repeat until happy!

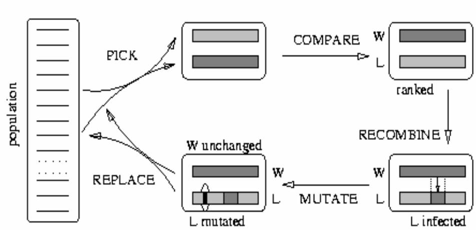
You needn't even have death !

microbes can evolve by **horizontal** transmission of genes (within the same generation) rather than (or as well as) **vertical** transmission (down the generations, from parents to offspring).

ie recombination happens **within** generations

Microbial sex =
'hey, wanna swap some of my genes for yours?'
rather than
'lets make babies'

Microbial Genetic Algorithm – the picture



Microbial Genetic Algorithm – the algorithm

✓Pick two genotypes at random
✓Compare scores -> Winner and Loser
✓Go along genotype, at each locus
–with some prob copy from Winner to Loser (overwrite)
–with some prob mutate that locus of the Loser

So **ONLY** the Loser gets changed
(gives a version of Elitism for free!)

This allows what is technically a one-liner GA (bar the evaluate(), which is problem-specific) -- quite a long line !

Microbial Genetic Algorithm – the one-liner

```
/* tournament loop */
for (t=0;t<END;t++)
        /*  loop along genotype of winner of tournament,
         selected in initial loop conditions */
        for (W=(evaluate(a=POP*drand48())>
                evaluate(b=POP*drand48()) ? a : b),
                        L=(W==a ? b : a), i=0; i<LEN; i++)
                /* throw dice to decide: cross or mutate */
                if ((r=drand48())<REC+MUT)
                /* update genotype of loser */
                gene[L][i]=(r<REC ? gene[W][i] : gene[L][i]^1);
```


… or slightly longer

```
int gene[POP][LEN];

Initialise genes at random;  define problem-specific evaluate(n)

/* tournament loop */
for (t=0;t<END;t++) {
        /*  pick 2 at random, find Winner and Loser */
        a=POP*drand48();
        do {b=POP*drand48()}
                while (a==b);  /*make sure a and b different */
        if (evaluate(a) > evaluate(b)) {W=a; L=b;}
        else {W=b; L=a;}
To be continued …
```

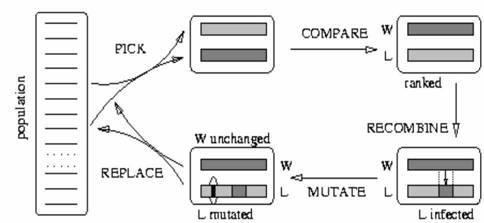

… continued

Continued …

```
        for (i=0;i<LEN;i++) {
                if (drand48()<REC) /* cross with probability REC */
                        gene[L][i]=gene[W][i];
                if (drand48()<MUT) /* mutate with probability MUT */
                        gene[L][i]=1-gene[L][i]; /* flip bit */
}       /* end tournament loop */
```

Possible values for **REC=0.5; ?** And **MUT=1.0/LEN; ?**

Microbial Genetic Algorithm – the picture



Is there a point ?

Microbial GA paper via my home page
http://www.informatics.susx.ac.uk/users/inmanh

It does actually work.

By no means guaranteed to be better than other GAs -- but does show **how really simple a GA can be**, and still work !

Apart from the one line, it needs declaration of gene[**POP**][**LEN**], initialisation of a random popn, and *evaluate(n)* that returns fitness of n^{th} member.

Embodied Evolution

Richard Watson, at Brandeis (papers available on web) has modified this to use with real robots in '**Embodied Evolution**'.

Robots go around 'broadcasting' their genes, and listening out to other broadcasts.

Fitter robots 'shout louder' (or more often)

Weaker robots are more likely to listen in, and use the genes they 'hear' to copy over their own.

A mini-GA project – for Seminars week 3

EASy

You have 10 cards numbered 1 to 10.

You have to divide them into 2 piles so that:

1) The **sum** of the first pile is as close as possible to 36
2) **And** the **product** of all in second pile is as close as poss to 360

Hint: call the piles **'0'** and **''1'**, and use binary genotypes of length 10 to encode any possible solution.

Think of a suitable fitness function.

Any questions so far … ?

EASy

… with luck there should be time left for GA-related questions.

Non-Symbolic AI lecture 4

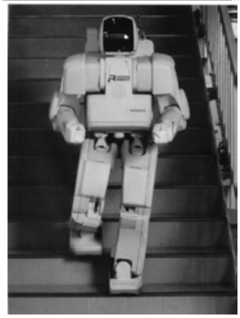
A major difference between Symbolic and Non-Symbolic AI approaches is in modelling, or emulating, Cognition or control – in artificially intelligent machines such as robots.

Symbolic, or Classical, AI tended to think in terms of control being focussed within a central, reasoning brain.

Given a task (for a human or a robot) such as ‘open the door’ or ‘catch the ball’, Symbolic AI assumes that the task can be turned into a set of propositions, using probably logic and maths.

Then this is now a ‘problem to be solved’ using the brain as a computer (… or the computer as a brain !)

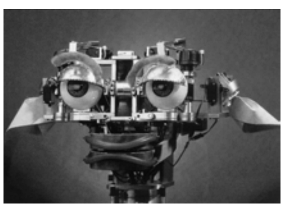
Robotics is used for …



… publicising the technical expertise of car companies – the Honda robot

Robotics is used for …

… working out how expressions communicate emotions



Robotics is used for …

… toys

Robotics is used for …

… and for science

-- as a way of understanding how animals and humans work by trying to build artificial ones.

Artificial Life.

Creating Robots in Man’s Image

Whether or not God created Man in His image, it is inevitably the case that Man and Woman create robots in their image.

Puppets, revealing how we (… those in the robot/cognitive science or philosophy business) really think of ourselves.

Doing ‘Philosophy of Mind’ with robots has one enormous disadvantage over conventional philosophy … …

… … you cannot fudge things, or appeal to magic!

Brains and Bodies

There is a traditional view that all the intelligence of a creature is in some rational brain – maybe like a computer – and the body is just ‘an afterthought’.

Here is an 8-legged walking robot like this – with an “artificially evolved brain” sitting inside the onboard computer.

Cognition

21st Century scientific human cognition
is different from that of
humans 3000 years ago
is different from that of
our ancestors of 2 billion years ago
is different from that of
our descendants of 2 billion years later
(... if there will be any ...)

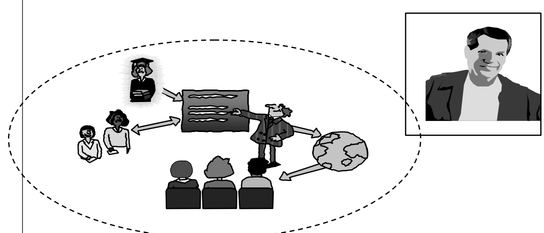
Descartes

Much of classical AI can be traced back to Descartes (early 17thC)

Dualism -- the separation of the mental and the physical. Cartesian objectivity:

~~"there just is a way the world is, independent of any observer. The scientist is a spectator from outside, a God's-eye view"~~

The view from outside



"The world is physical, knowledge is mental (something different)"

Classical AI

When building robots, this gives Classical AI approach where the robot is a scientist-spectator, seeking information from outside.

"SMPA" -- so-called by Brooks (1999)

- S sense
- M model
- P plan
- A action

Computing a model

The model is 'computed' from the sensory inputs.

But what is the computer metaphor?

The Computer metaphor

A Turing machine is a formal way of carrying out an algorithm -- a list of explicit instructions.

BUT beware of a simple confusion:-

When the astronomer calculates where the moon will be at 12:00 noon on May 1st, she carries out computations. She is a scientist-spectator.

But the moon does not carry out computations -- it 'just moves' in a deterministic way.

Classical AI confusion

The Classical AI approach tends to confuse these two -- tends to (mistakenly) think that "the ~~brain does computations~~".

To clarify: we can use a computer to simulate (predict) the movement of the moon -- even to control a model planetary system.

Similarly we can use a computer to simulate (predict) the dynamics of a nervous system -- even to control a robot with a model 'brain'

-- but this does not mean that the "brain computes" !

'Reasoning all the way down'

The Classical AI approach, obsessed with reasoning and computing, assumed that ***even*** something as simple as walking across the room, maintaining one's balance, required reasoning and computation … …

… … "Sense Model Plan Action" …

… … Brain controlling muscles

But look at this ---

Passive Dynamic Walking

'Natural walking behaviour', stable to small perturbations, can emerge from 'all body and no brain' !
It is the dynamics that count, whether the dynamics arise with or without a coupled nervous system.

Dan Jung's walker movie www.msc.cornell.edu/~ruinalab/pdw.html
"Passive Dynamic Walking", from Tad McGeer

Walking without a nervous system

Alternatives to the Classical Approach

There are different philosophical perspectives such as those of Heidegger / Merleau-Ponty / Wittgenstein
that might affect the way in which one designs robots.

These are difficult people to read, and they say little or nothing about robots !

Nevertheless, they offer a different perspective which has recently been crucially important

Other sources

Brooks 1999. Cambrian Intelligence
Dreyfus 1972. What Computers Can't Do
Winograd and Flores 1986.
    Understanding Computers and Cognition.
Pfeifer and Scheier 1999.
    Understanding Intelligence
Maturana and Varela 1987.
    The Tree of Knowledge

Situatedness and Embodiment

The Dynamical Systems view of Cognition

Heidegger …

…rejects the simplistic objective view, that the "~~objective physical world is the primary reality~~ that we can ~~be sure of~~"

He also rejects the opposite idealistic/subjective view that ~~"our thoughts are the primary reality"~~

The primary reality is our everyday practical lived experience, as we reach for the coffee or switch on the light

This is more fundamental than detached theoretical reflection.

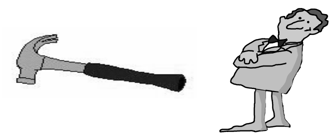
Reasoning only came later …

This actually makes sense from a Darwinian evolutionary perspective (though Heidegger would not say this)

-- our human language / reasoning powers arrived only 'recently' (last few 10,000 years, 100,000s ?)

From a phylogenetic and ontogenetic view,
we are organisms/animals first
    -- thinking humans only later.

What comes first?



Our unreflective tool-using is primary
    -- only when something goes wrong do we need to switch into 'reflective' mode.

Any lessons for robotics?

This is true (Wittgenstein suggests) even for language skills:

"In general we don't *use* language according to strict rules -- it hasn't been taught us by means of strict rules either"

What lessons for robots from these alternative views? At first sight, they are negative and unhelpful !

For everyday robot actions this implies we should do without planning, without the computational model, without internal representations ... . but what should we do instead ?

Dynamic skills all the way up?

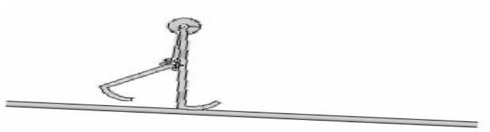
Perhaps rather than 'Reasoning all the way down' …

… we should think in terms of 'Dynamic skills all the way up'

Two initial lessons -- cognition is

EASy

- Situated: a robot or human is always already in some situation, rather than observing from outside.
- Embodied: a robot or human is a perceiving body, rather than a disembodied intelligence that happens to have sensors.



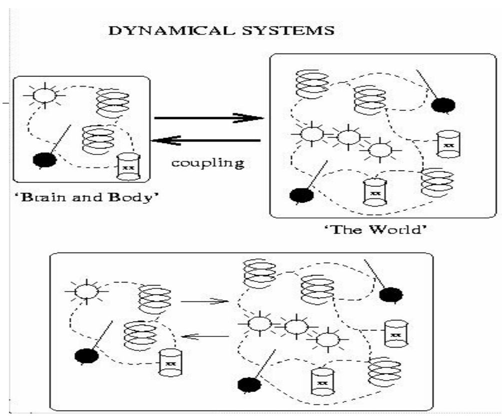
The Dynamical Systems view of Cognition

EASy

...animals are endowed with nervous systems whose dynamics are such that, when coupled with the dynamics of their bodies and environments, these animals can engage in the patterns of behavior necessary for their survival"

Beer & Gallagher 1992.

EASy



A Crucial Difference

EASy

What is one crucial difference between the Classical AI approach and the Dynamical Systems approach ?

Classical AI and computational approaches do not take account of time --
'life as a series of snapshots

Dynamical Systems approach --
time is central, 'life as process'

EASy

How can you design Dynamical Nervous Systems?

Brooks' Subsumption architecture is one way.

Evolutionary Robotics is another.

(Something crudely like the way we humans were designed !)

Subsumption architecture (1)

EASy

Fig. 1. Traditional decomposition of a mobile robot control system into functional modules.

Fig. 2. Decomposition of a mobile robot control system based on task-achieving behaviors.

(1a)

```
                    P    M    P    T    M
                    E    O    L    A    O
                    R    D    A    S    T
                    C    E    N    K    O
SENSORS   >>>>>     E >>> L >>> N >>>    >>> R   >>>>>   ACTUATORS
                    P    L    I    E
                    T    I    N    X    C
                    I    N    G    E    O
                    O    G         C    N
                    N              U    T
                                   T    R
                                   E    O
                                        L
```

Traditional decomposition of a mobile robot control system into functional modules

Brooks' alternative

Brooks' alternative is in terms of many individual and largely separate **behaviours** – where any one behaviour is generated by a pathway in the 'brain' or control system all the way from Sensors to Motors.

No Central Model, or Central Planning system.

(1b)

```
                    REASON ABOUT BEHAVIOR OF OBJECTS
                    PLAN CHANGES TO THE WORLD
                    IDENTIFY SUBJECTS
SENSORS  >>>>>>     MONITOR CHANGES                    >>>> ACTUATORS
                    BUILD MAPS
                    EXPLORE
                    WANDER
                    AVOID OBJECTS
```

Decomposition of a mobile robot control system based on task-achieving behaviors

Subsumption architecture (2)

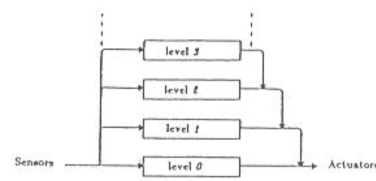
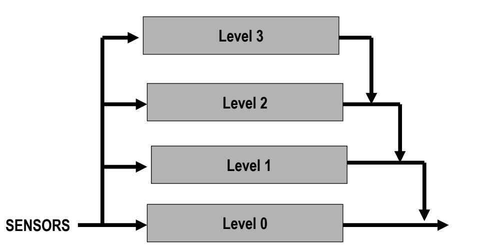


Fig. 3. Control is layered with higher level layers subsuming the roles of lower level layers when they wish to take control. The system can be partitioned at any level, and the layers below form a complete operational control system.

(2a)



Control is layered with higher levels subsuming control of lower layers when they wish to take control.

Subsuming

'Subsume' means to take over or replace the output from a 'lower layer'.

The 2 kinds of interactions between layers are

1. Subsuming
2. Inhibiting

Generally only 'higher' layers interfere with lower, and to a relatively small extent – this assists with an incremental design approach.

Non-Symbolic AI lecture 5

We shall look at 2 alternative non-symbolic AI approaches to robotics

- ❑Subsumption Architecture
- ❑Evolutionary Robotics

Classical AI

When building robots, the Classical AI approach has the robot as a scientist-spectator, seeking information from outside.

"SMPA" -- so-called by Brooks (1999)

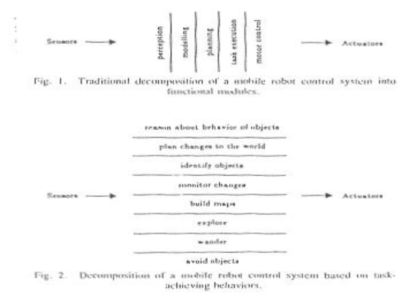
- S sense
- M model
- P plan
- A action

Brooks' alternative

Brooks' alternative is in terms of many individual and largely separate **behaviours** – where any one behaviour is generated by a pathway in the 'brain' or control system all the way from Sensors to Motors.

No Central Model, or Central Planning system.

Subsumption architecture (1)



(1a)

```
                    P    M    P    T    M
                    E    O    L    A    O
                    R    D    A    S    T
                    C    E    N    K    O
SENSORS   >>>>>     E >>> L >>> N >>>    >>> R   >>>>>   ACTUATORS
                    P    L    I    E
                    T    I    N    X    C
                    I    N    G    E    O
                    O    G         C    N
                    N              U    T
                                   T    R
                                   E    O
                                        L
```

Traditional decomposition of a mobile robot control system into functional modules

(1b)

```
                        REASON ABOUT BEHAVIOR OF OBJECTS
                        PLAN CHANGES TO THE WORLD
                        IDENTIFY SUBJECTS
SENSORS  >>>>>>         MONITOR CHANGES                  >>>>  ACTUATORS
                        BUILD MAPS
                        EXPLORE
                        WANDER
                        AVOID OBJECTS
```

Decomposition of a mobile robot control system based on task-achieving behaviors

Subsumption architecture (2)

EASy

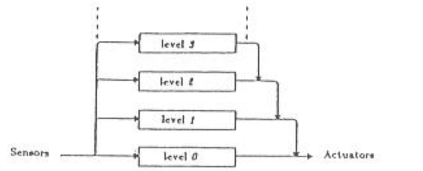
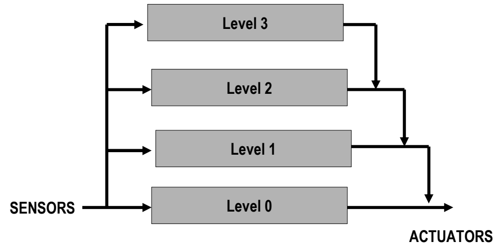


Fig. 3. Control is layered with higher level layers subsuming the roles of lower level layers when they wish to take control. The system can be partitioned at any level, and the layers below form a complete operational control system.

(2a)

EASy



Control is layered with higher levels subsuming control of lower layers when they wish to take control.

Subsuming

EASy

'Subsume' means to take over or replace the output from a 'lower layer'.

The 2 kinds of interactions between layers are

1. Subsuming
2. Inhibiting

Generally only 'higher' layers interfere with lower, and to a relatively small extent – this assists with an incremental design approach.

Subsumption architecture (3)

EASy

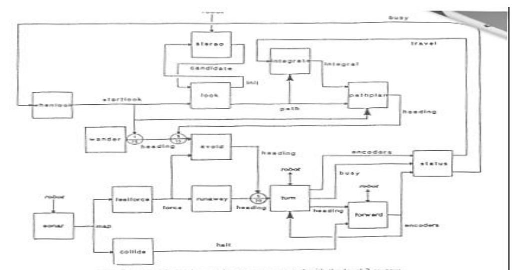


Fig. 7. Level 0 and 1 control systems augmented with the level 2 system.

Subsumption architecture (4)

EASy

That looked a bit like a Network – except rather than (artificial) Neurons the components are versions of

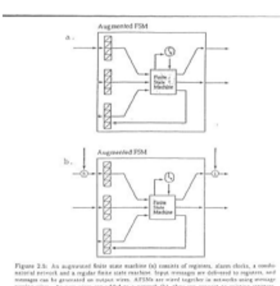
AFSMs

Augmented

Finite

State

Machines



AFSMs

EASy

An AFSM consists of registers, alarm clocks (**time!**), a combinatorial network and a regular finite state machine. Input messages are delivered to registers, and messages can be generated on output wires.

As new wires are added to a network (lower figure before), they can connect to existing registers, inhibit outputs, or suppress inputs.

Herbert

EASy

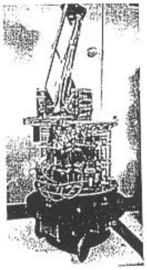


Figure 12.2: Herbert was controlled by a "colony" of independent agents all wanting to con
sensing to action with no communication with eachother except via the world.

16 infrared sensors, compass, laser light striper for finding soda-cans. 24 8-bit microprocessors distributed around the body

Herbert's actions

EASy

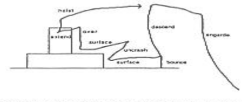


Figure 12.13: Like figure 12.12 this shows the path of the finger tips while searching for a soda can. This time there is an obstacle on the table surface, and a very different "plan" emerges from the interaction of the robot and its environment.

SKIM level of control

Figure 12.10: The strategy for Herbert's arm to find something that is in front of it is for it to skim along a surface in a sawtooth pattern. It reaches forward and down, bouncing up whenever the touch sensors on the finger tips detect a surface.

Subsumption summary

EASy

- ❑New philosophy of hand design of robot control systems
- ❑Incremental engineering – debug simpler versions first
- ❑Robots must work in **real time** in the **real world**
- ❑Spaghetti-like systems unclear for analysis
- ❑Not clear if behaviours can be re-used
- ❑Scaling – can it go more than 12 behaviours?

Evolutionary Robotics

EASy

Evolutionary Robotics (ER) can be done
✓for Engineering purposes - to build useful robots
✓for Scientific purposes - to test scientific theories

It can be done
✓for Real or
✓in Simulation

Here we shall start with the most difficult, robots with Dynamic Recurrent Neural Nets, tested for Real.

Then we shall look at simplifications and simulations.

The Evolutionary Approach

EASy

Humans are highly complex, descended over 4 bn yrs from the 'origin of life'.

Let's start with the simple first - 'today the earwig'
(not that earwigs are that simple ...)

Brooks' subsumption architecture approach to robotics is 'design-by-hand', but still inspired by an incremental, evolutionary approach:

✓Get something simple working (debugged) first
✓Then try and add extra 'behaviours'

What Class of 'Nervous System'

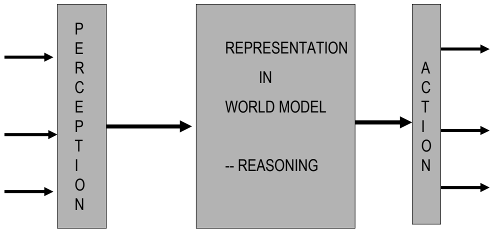
EASy

When evolving robot 'nervous systems' with some form of GA, then the genotype ('artificial DNA') will have to encode:

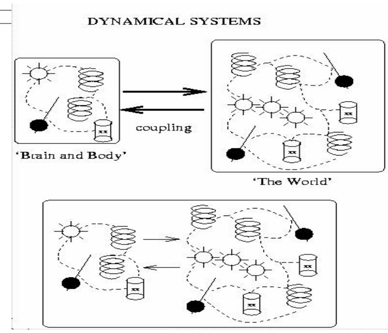
✓The architecture of the robot control system
✓Also maybe some aspects of its body/motors/sensors

But what kind of robot control system, what class of possible systems should evolution be 'searching through' ?

… could be a classical approach ?



… or a Dynamical Systems Approach



DS approach to Cognition

cf R Beer 'A Dynamical Systems Perspective on Autonomous Agents' Tech Report CES-92-11. Case Western Reserve Univ. Also papers by Tim van Gelder.

In contrast to Classical AI, computational approach, the DS approach is one of 'getting the dynamics of the robot nervous system right', so that (coupled to the robot body and environment) the behaviour is adaptive.

Brook's subsumption architecture, with AFSMs (Augmented Finite State Machines) is one way of doing this.

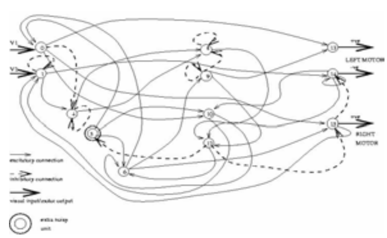
Dynamic Recurrent Neural Networks

DRNNs (or CTRNs = Continuous Time Recurrent Networks) are another (really quite similar way).

You will learn about other flavours of Artificial Neural Networks (ANNs) in Adaptive Systems course.
-- eg ANNs that 'learn' and can be 'trained'.

These DRNNs are basically different -- indeed basically just a convenient way of specifying a class of dynamical systems
-- so that different genotypes will specify different DSs, giving robots different behaviours.

One possible DRNN, wired up

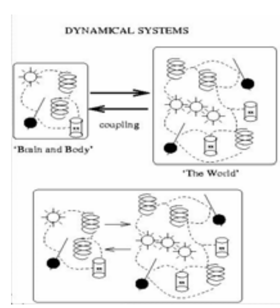


This is just ONE possible DRNN, which ONE specific genotype specified.

Think of it as …

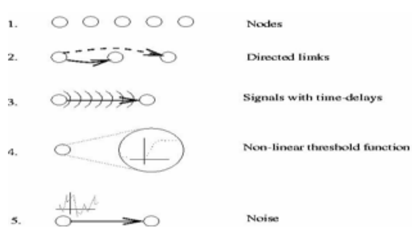
Think of this as a nervous system with its own Dynamics.

Even if it was not connected up to the environment
(I.e. it was a 'brain-in-a-vat'), it would have its own dynamics, through internal noise and recurrent connections)



DRNN Basics

The basic components of a DRNN are these
(1 to 4 definite, 5 optional)



ER basics

The genotype of a robot specifies
(through the encoding genotype->phenotype that WE decide on as appropriate)
how to 'wire these components up' into a network connected to sensors and motors.

(Just as there are many flavours of feedforward ANNs, there are many possible versions of DRNNs – in a moment you will see just one.)

Then you hook all this up to a robot and evaluate it on a task.

Evaluating a robot

When you evaluate each robot genotype, you
✓Decode it into the network architecture and parameters
✓Possibly decode part into
body/sensor/motor parameters
✓Create the specified robot
✓Put it into the test environment
✓Run it for n seconds, scoring it on the task.

Any evolutionary approach needs a selection process, whereby the different members of the population have different chances of producing offspring according to their **fitness**

Robot evaluation

(Beware - set conditions carefully!)

Eg: for a robot to move, avoiding obstacles -- have a number of obstacles in the environment, and evaluate it on how far it moves forwards.

Have a number of trials from random starting positions
✓ take the average score, **or**
✓ take the worst of 4 trials, **or**
✓ (alternatives with different implications)

Deciding on appropriate fitness functions can be difficult.

DSs -> Behaviour

The genotype specifies a DS for the nervous system

Given the robot body, the environment, this constrains the behaviour

The robot is evaluated on the behaviour.

The phenotype is (perhaps):

✓the architecture of the nervous system(/body)
✓or ... the behaviour
✓or even ... the fitness

Robustness and Noise

For robust behaviours, despite uncertain circumstances, noisy trials are neeeded.

Internal noise (deliberately put into the network) affects the dynamics (eg self-initiating feedback loops) and (it can be argued) makes 'evolution easier'
-- 'smooths the fitness landscape'.

Summarising DSs for Robot Brains

EASy

They have to have **temporal** dynamics.
Three (and there are more...) possibilities are:

(1) Brook's subsumption architecture

(2) DRNNs as covered in previous slides

(3) Another option to mention here: Beer's networks

see Beer ref. cited earlier, or "Computational and Dynamical Languages for Autonomous Agents", in Mind as Motion, T van Gelder & R. Port (eds) MIT Press

Beer's Equations

EASy

Beer uses **CTRNNs** (continuous-time recurrent NNs), where for each node (i = 1 to n) in the network the following equation holds:

$$\tau_i \frac{dy_i}{dt} = -y_i + \sum_{j=1}^{n} w_{ji}\sigma(y_j - \theta_j) + I_i(t)$$

y_i = activation of node i
τ_i = time constant, w_{ji} = weight on connection from node j to node i
$\rho(x)$ = sigmoidal = $(1/1+e^{-x})$
η_i= bias,
I_i = possible sensory input.

Applying this for real

EASy

One issue to be faced is:
- Evaluate on a real robot, or
- Use a Simulation ?

On a real robot it is expensive, time-consuming -- and for evolution you need many many evaluations.

Problems of simulations

EASy

On a simulation it should be much faster
(though note -- may not be true for vision)
cheaper, can be left unattended.

BUT AI (and indeed Alife) has a history of toy, unvalidated simulations, that 'assume away' all the genuine problems that must be faced.

Eg: grid worlds "move one step North"

Magic sensors "perceive food"

Principled Simulations ?

EASy

How do you know whether you have included all that is necessary in a simulation?

-- only ultimate test, **validation,** is whether what works in simulation ALSO works on a real robot.

How can one best insure this, for Evolutionary Robotics ?

'Envelope of Noise' ?

EASy

Hypothesis: -- "if the simulation attempts to model the real world fairly accurately, but where in doubt extra noise (through variations driven by random numbers) is put in, then evolution-in-a-noisy-simulation will be more arduous than evolution-in-the-real-world"

Ie put an envelope-of-noise, with sufficient margins, around crucial parameters whose real values you are unsure of.

"Evolve for **more robustness** than strictly necessary"

Problem: some systems evolved to rely on the existence of noise that wasnt actually present in real world!

Jakobi's Minimal Simulations

EASy

See, by Nick Jakobi:

(1) Evolutionary Robotics and the Radical Envelope of Noise Hypothesis and
(2) The Minimal Simulation Approach To Evolutionary Robotics

available on http://www.cogs.susx.ac.uk/users/nickja/

Minimal simulation approach developed explicitly for ER -- the problem is often more in simulating the environment than the robot.

Minimal Simulation principles

EASy

Work out the minimal set of environmental features needed for the job -- the **base set**.

Model these, with some principled envelope-of-noise, so that what uses these features in simulation will work in real world
-- '**base-set-robust**'

Model everything ELSE in the simulation with wild, unreliable noise -
- so that robots cannot evolve in simulation to use anything other trhan the base set
-- '**base-set-exclusive**'

Non-Symbolic AI lecture 6

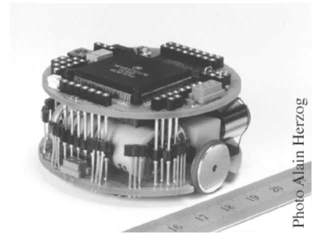
Last lecture we looked at non-symbolic approaches to robotics – new methods of approaching **Engineering** problems

This lecture we look at simulated agents as a Non-symbolic AI (or Alife) way of asking and answering scientific issues.

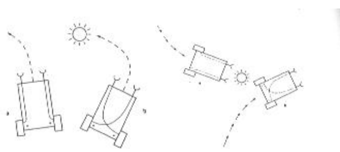
Braitenberg vehicles

Braitenberg vehicles have sensors (typically few, and simple) and motors (typically 2, driving left and right wheels independently).

A bit like Khepera robots



Braitenberg (2)



Then very simple direct connections between sensors and motors can, for example, produce light-seeking or light-avoiding behaviour.

Above examples may have direct linear connections, plus some bias term on the motors so as to move even when no sensory input.

More complex Braitenberg vehicles

These are feedforward ANNs, feeding from sensory inputs to motor outputs. They can be made more complex (while remaining feedforward) by

- Adding bias terms
- Adding internal nodes – multilayer ANN
- Adding non-linearities (… threshold functions, sigmoids)
- Adding extra sensors for new senses
- … etc etc

Reactive Behaviour

But as long as a Braitenberg vehicle has a feedforward ANN (however many sensors, however many layers, however non-linear) from sensors to motors, then it is merely a **Reactive System.**

Warning: different people use the term 'reactive' in different ways, here I am explicitly using it to mean 'no internal state'. If the robot reacts to the same inputs the same way on Mon, Tues, Wed …, then it is reactive under my definition.

A coke machine that requires 2 coins to deliver a can is *not* reactive – since reactions to inserting a coin differ.

Non-reactive behaviour

Subsumption architecture produces non-reactive behaviour (remember the 'alarm clocks' in AFSMs)

DRNNs produce non-reactive behaviour.

All really complex behaviour is non-reactive (in this sense) – even though Braitenberg vehicles demonstrate how surprisingly interesting one can get merely with reactive.

Non-reactive means that behaviour changes according to previous experience – which brings up notions such as **memory** and **learning**.

What is Learning ?

EASy

Learning is a **behaviour** of an organism – animal or human or robot, or piece of software behaving like these.

More exactly, it is a **change of behaviour** over time, so as to improve performance at some task.

On Monday I could not ride a bicycle – my behaviour was 'falling-off-the-bike'.

By Friday my behaviour had changed to 'successfully riding the bike'.

Strictly, ANN weight changes are **plasticity**, not learning – though the whole system may learn via this plasticity.

Evolution and Learning

EASy

Exploring Adaptive Agency II: Simulating the Evolution of Associative Learning
Peter Todd and Geoffrey Miller, pp. 306-315 in
From Animals to Animats, J-A Meyer & S. Wilson (eds)
MIT Press 1991 (Proc of SAB90)

Looking at some aspects of the relationship between *evolution* and *learning*.

Why bother to Learn - 1?

EASy

Todd & Miller suggest 2 reasons:

(1) Learning is a cheap way of getting complex behaviours, which would be rather expensive if 'hard-wired' by evolution.
Eg: parental imprinting in birds –

'the first large moving thing you see is mum, learn to recognise her'.

Why bother to Learn – 2?

EASy

(2) Learning can track environmental changes faster than evolution.

Eg: it might take millennia for humans to evolve so as to speak English at birth, and would require the English language not to change too much.

But we have evolved to learn whatever language we are exposed to as children -- hence have no problems keeping up with language changes.

Learning needs Feedback

EASy

Several kinds of feedback:
Supervised -- teacher tells you exactly what you should have done
Unsupervised -- you just get told good/bad, but not what was wrong or how to improve.
(..cold...warm...warmer...)

Evolution roughly equates to unsupervised learning
-- if a creature dies early then this is negative feedback as far as its chances of 'passing on its genes' are concerned -- but evolution doesn't 'suggest what it *should* have done'.

Evolving your own feedback?

EASy

However, it may be possible, under some circumstances, for evolution to create, within 'one part of an organism', some subsystem that can act as a 'supervisor' for another subsystem.

Cf.DH Ackley & ML Littman.
Interactions between learning and evolution. In *Artificial Life II*, Langton et al (eds), Addison Wesley 1991.

Later work by same authors on
Evolutionary Reinforcement Learning

The Todd & Miller model

Creatures come across food (+10 pts) and poison (-10)

Food and poison always have different *smells,* sweet and sour. **BUT** sometimes smells drift around, and cannot be reliably distinguished. In different worlds, the reliability of smell is x% where 50 <= x <= 100.

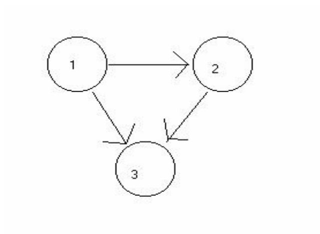
Food and poison always have different *colours,* red and green. **BUT** in some worlds it is food-red poison-green, in other worlds it is food-green poison-red. The creatures' vision is always perfect, but 'they dont know whether red is safe or dangerous'.

The model – ctd.

Maybe they can learn, using their unreliable smell?
Todd & Miller claim that simplest associative learning needs:
(A) an input that (unreliably) senses what is known to be good or bad (smell)
(B) another sensor such as that for colour, above
(C) output such that behaviour alters fitness.
(D) an evolved, fixed connection (A)->(C) with the appropriate weighting (+ve for good, -ve for bad)
(E) a learnable, plastic connection (B)->(C) which can be built up by association with the activation of (C)

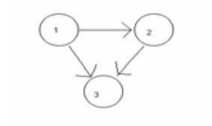
Their 'Brains'

Creatures brains are genetically specified, with exactly 3 neurons connected thus:



Genetic specification

The genotype specifies for each neuron whether it is
- ✓input sweet-sensor
- ✓input sour-sensor
- ✓input red-sensor
- ✓input green-sensor
- ✓hidden unit or interneuron
- ✓output or decision unit: eat/dont-eat



- ✓and for each neuron the bias (0, 1, 2, 3) (+/-)
- ✓and for each link the weight (0, 1, 2, 3) (+/-)
- ✓and whether weight is fixed or plastic

Plastic links

Some of the links between neurons are fixed, some are plastic.

For plastic weights on a link from P to Q, Hebb rule:
Change in WEIGHTPQ $= k * A_P * A_Q$
where A_P is the current activation of neuron P.

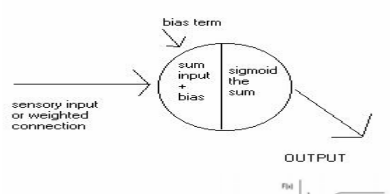
I.e.:- if the before and after activations are the same sign (tend to be correlated), increase strength of link
If opposite sign (anti-correlated), decrease strength

(n.b.)

(Actually, if you check the details of the Todd and Miller paper, it is a bit more complicated.

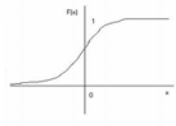
The Hebb rule assumes outputs can be positive or negative. The sigmoids used (see a couple of slides later) are always positive, but these are then rescaled to allow outputs of hidden and motor units to fall within range –1 to +1. So we are OK for the Hebb rule to make sense.)

Within each neuron



The sigmoid is a 'squashing function' such as

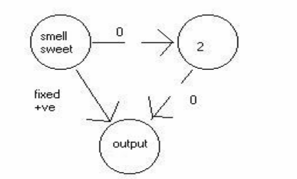
$$f(x) = \frac{1}{1+e^{-x}}$$



Possible brains

So one possible genetically designed brain would be this: **colour-blind eater** – this is **not** a learner.

Whatever neuron 2 is, the links are 0, hence it can be ignored.
This depends purely on smell, and has the connection wired up with the right +ve sign, so that it eats things that smell sweet.

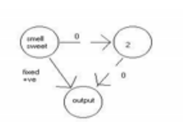


Is a colour-blind eater any good?

This creature depends purely on smell, and has the connection wired up with the right +ve sign, so that it eats things that smell sweet.

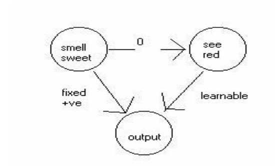
Though it may occasionally ignore food that (noisily) smells sour, and eat poison that (noisily) smells sweet, on average it will do better than random.

So evolving within a population, this (non-learning) design will do better than random, and increase in the population.



A learning brain

This different possible design is a **colour-learner**



If this one is born in a world where smell is 75% accurate, and food is red, then (more often than not) seeing red is associated with the positive output.

Is a colour-learner any good?

… seeing red is associated with the positive output. So, with Hebbs rule, the RH connection gets built up +ve more strongly, until it fires the output on its own -- it can even over-rule the smell input on the 25% of occasions when it is mistaken.

Contrariwise, if it is born in a world where food is green, then Hebbs rule will build up a strong –ve connection -- with the same results.

So over a period, this will do better than the previous one

Evolutionary runs

Populations are initialised randomly.

Of course many have no input neurons, or no output neurons, or stupid links
- these will behave stupidly or not at all.

The noise level on smell is set at some fixed value between 50% accurate (chance) and 100% accurate.

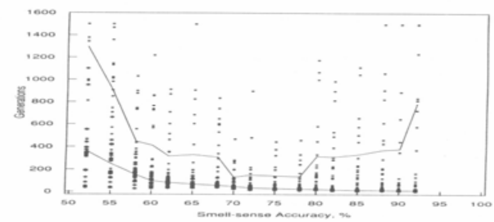
Then individuals are tested in a number of worlds where (50/50) red is food or red is poison.

Keeping statistics

Statistics are kept for how long it takes for **colour-blind eater** to turn up and take over the population; and for **colour-learner** likewise.
[Statistics: multiple runs to reduce chance elements, keep record of standard deviations]

❑When smell noise-level is 50%, then there is no available information, no improvement seen.
❑When smell is 100% accurate, then colour-learning is unnecessary.
❑What happens between 50% and 100% ?

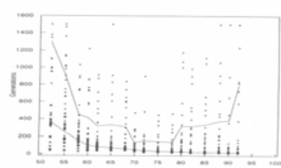
Results on a graph



Colour-learner= top line colour-blind eater=bottom line

U-shaped curve

As smell-sense accuracy goes from 50% (left) to 100% (right), average number of generations needed to evolve **colour-blind eater** decreases steadily (lower line)



But average number of generations needed to evolve **colour-learner** goes through the U-shaped curve (upper line)

Baldwin Effect

A much more subtle and interesting interaction between Learning and Evolution is the Baldwin effect -- named after Baldwin (1896).

Roughly speaking, this is an effect such that, under some circumstances, the ability of creatures to learn something guides evolution, such that in later generations their descendants can do the same job innately, without the need to learn.

Is it Lamarckian?

This sounds like Lamarckism -- "giraffes stretched their necks to reach higher trees, and the increased neck-length in the adults was directly inherited by their children".

Lamarckism of this type is almost universally considered impossible -- why ??

The Baldwin effect gives the impression of Lamarckism, without the flaws.
Be warned, this is tricky stuff !

Non-Symbolic AI lecture 7

Different types of ANNs for different jobs.

So far we have looked primarily at ANNs for robot control, varying from simple feedforward for simple Braitenberg vehicles (for reactive behaviour, in the sense of no internal memory)

… to simple Hebbian plasticity ('learning') for exploring the relationship between Learning and Evolution

… to more complex recurrent networks with **time** involved –

- E.g. subsumption architecure considered as a kind of ANN
- Or Dynamic Recurrent NNs

Pattern recognition

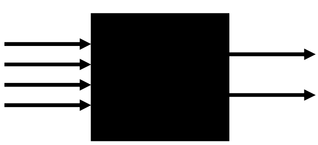
A lot – probably by far the most – of ANNs used are **not** recurrent, are feedforward with no timing issues involved, and can be trained in various possible ways to learn (statistical)

input -> output relationships.

Let's recognise that these ANNs probably have near-zero relationship to what actually goes on with real neurons in the brain, and just consider them as potentially really useful pattern-recognisers – all sorts of practical applications.

Rapid review

Rapid review of the basics of feedforward ANNs -- which most of you should have covered already in Further AI and perhaps elsewhere.

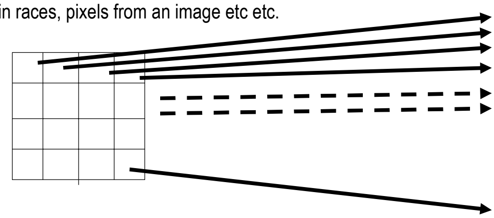


INPUTS **OUTPUTS**

Inputs and Outputs

Inputs are a set (or vector) of real numbers (or could be limited to eg just 0s and 1s).

Could be data from the stockmarket, past performance of horses in races, pixels from an image etc etc.



Inputs and Outputs

Outputs: there might be just one, or many outputs of real values (vector).

These outputs are, roughly, what a (properly trained) Black Box **predicts** from the Inputs.

E.g. what the Stockmarket index will be tomorrow, how fast the horse will run in the 2:30pm at Newmarket, is the picture like a dog (output 1 high) or a cat (output 2 high) or neither (if both outputs low)

Any specific Black Box implements a function from **In** to **Out**.

Out = BBf(**In**)

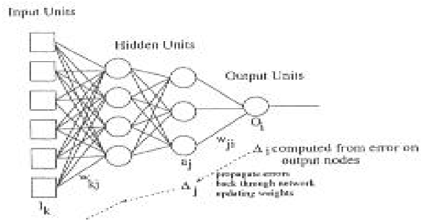
Training and Testing

If the Black Box is intended to be a dog-recogniser (eg 10x10=100 pixels input, 1 output which should be high for 'dog'), then ideally it should be testable with **all** possible input images, and output high only for the doggy ones.

There are zillions of possible input images. An ANN is one type of Black Box that can be trained on just a subset, a **training set** of typical doggy **and** non-doggy images.

Ideally it should then generalise to a **test set** of images it hasn't seen before

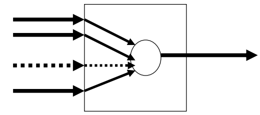
Inside the Black Box



Ultimately we will look at multi-layer ANNs, but let's start simple ..

Inside the Black Box

The simplest possible ANN with many inputs and one output.



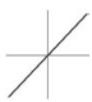
1 'neuron' inside the Black Box, weights on all the 100 inputs I, so the weighted inputs all get summed together at the node. If A is the activation of the node, then

$$A = \sum_{i=1}^{N} w_i I_i$$

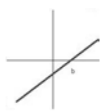
Transfer function

The output **Out** is going to be some function of the activation A. Simple possibilities include:

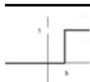
$$Out = A$$



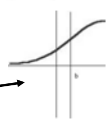
$$Out = w_o(A-b)$$



$$(A > b) \Rightarrow (Out = 1)$$
$$(A \le b) \Rightarrow (Out = 0)$$



$$Out = sigmoid(A-b) = \frac{1}{1+e^{-(A-b)}}$$



Linear or nonlinear

The first 2 are linear (the second has a bias term b, plus a weight).

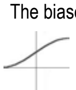
The next 2 are non-linear, including a sharp step-function or threshold function, and a smoother sigmoid.

(Step-function useful if eg Out=1 => dog, Out=0 =>not-dog !!)

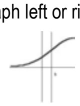
You can do far more complex pattern-recognition with non-linear functions. The sigmoid is a smooth, and differentiable, version of the step-function, and for practical reasons this turns out useful. So the sigmoid function is one to take note of.

Biases

The biases just shift the graph left or right.



$$Out = \frac{1}{1+e^{-A}}$$



$$Out = \frac{1}{1+e^{-(A-b)}}$$

But remember, A was just the weighted sum of inputs

$$A = \sum_{i=1}^{N} w_i I_i$$

Treat bias as another input

$$A = \sum_{i=1}^{N} w_i I_i$$

So if we pretend that there was another input, input value clamped to 1, with a weight of (-b), then we can treat it the same as the other inputs

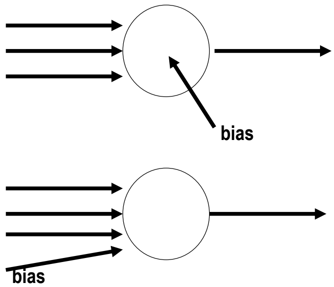
$$newA = oldA - b = \sum_{i=1}^{N} w_i I_i - b$$

$$= \sum_{i=1}^{N+1} w_i I_i$$

Where $w_{(n+1)} = -b$

And $I_{(n+1)} = 1$

Treating bias as another input



Treating bias as another input

So when you treat the bias (on any node in the network) as a weight on an input to that node whose input value is clamped to 1

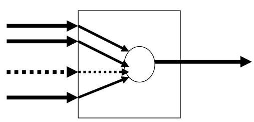
❑The equations and the programming come out a lot simpler

❑And the bias term can be ‘learnt’ by exactly the same method as all the other weights in the ANN are learnt, during training

❑So from now on we will assume that this trick is being used.

The simple Black Box

The simplest possible ANN with many inputs and one output.



Now we have started off looking at this simplest version, a single layer perceptron with 1 output.

Could be made a bit more complex if 2 or more outputs (is it a dog? Is it a cat?)

Learning Algorithm

We still haven’t even started to discuss any training method, whereby the appropriate weights (including biases) can be learnt through exposure to the training set

(eg lots of pictures of dogs, cats, other things, with the correct response known for each member of the training set).

Basically there are 2 classes of learning here (ignoring a third of ‘self-organisation’)

❑Reinforcement Learning

❑Supervised Learning

Jiggling the weights

Basically all these algorithms work on different versions of

❑Start off with random weights (and biases) in the ANN

❑Try one or more members of the training set, see how badly the outputs are compared to what they should be (compared to the target outputs)

❑Jiggle weights a bit, aimed at getting improvement on outputs

❑Now try with a new lot of the training set, or repeat again, jiggling weights each time

❑Keep repeating until you get quite accurate outputs

Reinforcement

In Reinforcement learning, during training an input (‘picture’) is presented to the Black Box, the Output (‘0.75 like a dog’) is compared to the correct output (‘1.0 of a dog’ !!) and the size of the error is used for training (‘wrong by 0.25’)

If there are 2 outputs (cats and dogs) then the total error is summed to give a single number (typically sum of squared errors). Eg “your total error on all outputs is 1.76”

Note that this just tells you how wrong you were, **not** in which direction you were wrong.

Like ‘Hunt the Thimble’ with clues of ‘warmer’ ‘colder’.

Supervised

In Supervised Learning the Black Box is given more information.

Not just ‘how wrong’ it was, but ‘in what direction it was wrong’

Like ‘Hunt the Thimble’ but where you are told ‘North a bit’ ‘West a bit’.

So you get, and use, far more information in Supervised Learning, and this is the normal form of ANN learning algorithm.

Reinforcement Learning vs Supervised

Genetic Algorithms are a form of Reinforcement learning.

So actually a GA is one perfectly good method of ‘evolving’ the weights of an ANN, whether it is 1-layer or multilayer.

Encode all the weights (and biases) on the genotype, use a population (randomly initialised), and use errors on the training set as the fitness function.

This is just one version of ‘jiggling the weights a bit’ – here it is mutation jiggling the weights.

You are, however, usually wasting information that can be used for Supervised Learning.

Perceptron Learning Algorithm

You should have covered this in Further AI. (copied from there)

Gradient descent trying to minimise error. For each training example, input I, expected target output T, actual output O.

Error E = T – 0

Jiggle each weight w_i by adding a term R x I_i x E, where R is a small constant called the *learning rate*.

This jiggles the weights in the right direction to decrease error, by an amount R which makes it a small jiggle.

Gradient descent.

The 1-layer algorithm

Initialise perceptron with a random set of weights

Repeat

for each training instance (I,T) do {

E = T – Out;

for (i=1i<=N;I++) {

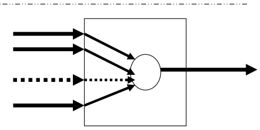
w[i] = w[i] + R * I_i * E;

}

} **until** error acceptably small.

What can the simple Perceptron do ?

The simplest possible ANN with many inputs and one output.



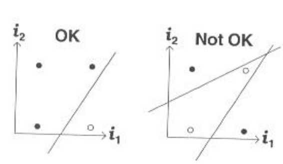
We are still looking at this very simple 1-layer perceptron, with 1 (or possibly more) outputs.

It can be proved (Perceptron Convergence Theorem) that **if** there is some set of weights that will do the pattern-recognition, or classification job we want, then the algorithm on previous slide will do the job.

However

However, it turned out that only relatively simple pattern-recognition, or classification, jobs can be done by the 1-layer perceptron – those that are ‘linearly separable’

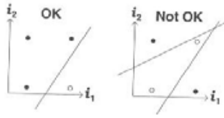
This is what Minsky & Paert’s 1969 book was all about – and this shot down ANNs for 2 decades ! Eg the XOR problem cannot



be tackled by such a perceptron

Linearly separable

This is a sketch of how a 2-input, 1-output perceptron needs to classify inputs.



It needs to distinguish black dots from open circles, in this training set of 4 examples.

In the left case, it can do so with a single straight line – and a 1-layer perceptron can handle this.

In the right case, it is not 'linearly separable', and cannot manage.

Extension to multi-layer perceptrons

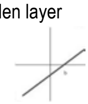
It turns out that we **can** in principle find Black Boxes that do such non-linear separation tasks **if**

- We have an extra 'hidden' layer
- We have a non-linear transfer function such as the sigmoid at the hidden layer
- The tricky bit – we can find a learning algorithm that copes with errors at the different layers, so as to jiggle all the weights appropriately
- Backpropagation was the algorithm that broke the logjam

Why the sigmoid ?

Suppose there was a linear transfer function at the hidden layer

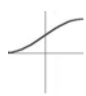
Then if you follow all the maths through, it turns out that effectively the hidden layer does not buy you anything extra – it is equivalent to just 1 layer



If it has to be non-linear, why not a step function?

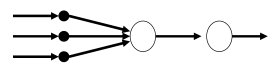
Turns out that backprop needs a smooth differentiable function, such as this:-

$$Out = \frac{1}{1+e^{-A}}$$



How big a hidden layer ?

If you just have 1 hidden node, then effectively you are back to a 1-layer ANN



You need at least 2, and roughly 'the more complex the classification task, the more hidden nodes you need'.

In principle, absolutely **any** continuous classification task can be done provided you have enough hidden nodes.

But you should not have too many, because of worries about overfitting.

Overfitting

If you have lots of hidden nodes, then you will have lots of weights (and biases) to learn.

Suppose you only have 10 members in your training set, but more than 100 weights, then learning will probably do the equivalent of memorising the *idiosyncracies* of the input/output pairs – and will not generalise sensibly to new inputs it hasn't seen before.

You can check for overfitting by keeping a few examples back, and after training seeing how well the Black Box generalises to this new test set.

Warning on Overfitting – When to worry/not worry

If you are training on a subset of all possible example patterns, this is when to worry about overfitting – because overtraining can fixate on the accidental biases of the training set.

BUT sometimes you could be training on the WHOLE possible set of examples (eg test problem for seminars in week 4) – then there is no possible overfitting to worry about.

So how many hidden nodes, then?

EASy

Ideally, just enough !!

There are (difficult) theoretical answers to this, but one approach is to try different numbers, and see how well the trained ANN generalises to an unseen test set in each case. Pick the best value.

In practice, one picks some number bu guesswork, experience, asking a friend – and if it works you stick with it, otherwise change!

Summary so far

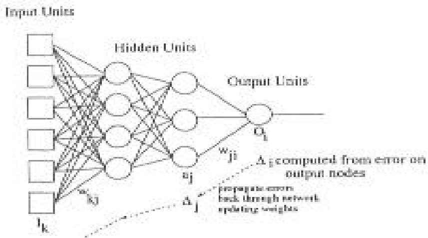
EASy

OK, next lecture we will go through the details of back-propagation, but a lot of the lessons have been already given.

- Weights and biases can be treated the same way
- We are going to use errors (output – Target) to jiggle the weights around till error decreases
- Reinforcement learning (GAs) is one possibility
- Supervised learning uses more information
- Present training set, use errors to jiggle weights

Non-Symbolic AI lecture 8

Backpropagation in a multi-layer perceptron



Jiggling the weights in each layer

When we present the training sets, for each Input we have an actual Output, compared with the Target gives the Error

Error E = T – 0

Backprop allows you to use this error-at-output to adjust the weights arriving at the output layer

… but then also allows you to calculate the effective error '1 layer back', and use this to adjust the weights arriving there

… and so on **back-propagating** errors through any number of layers

Differentiable sigmoid

The trick is the use of a sigmoid as the non-linear transfer function

$$y = g(x) = \frac{1}{1+e^{-x}}$$

Because this is nicely differentiable – it so happens that

$$\frac{dg}{dx} = g'(x) = g(x)(1-g(x))$$

How to do it in practice

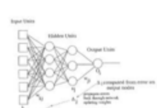
For the details, consult a suitable textbook

e.g. Hertx, Krogh and Palmer "Intro to the Theory of Neural Computation" Addison Wesley 1991

But here is a step by step description of how you have to code it up.

Initial decisions

Decide how many inputs you have. Add one more pseudo-input (clamped to value 1) for biases in next layer.



Decide how many hidden nodes you have. Add one more, clamped to value 1, for biases to next layer.

Probably you just have 1 hidden layer, tho in principle can be more.

Decide how many output nodes you have.

Weights

Now you can make vectors holding all the weights. Suppose there are a input units, b hidden units, c output units

float i_to_h_wts[a+1][b];

float h_to_o_wts[b+1][c];

The +1 in each case is to account for the biases.

Initialise all the weights to small random values.

Presenting the inputs

EASy

Now we are going to present members of the training set to the network, and see what outputs result – and how good they are.

We could present a whole **batch** from the training set, and calculate how to jiggle the weights on the basis of all the errors

Or we can do it incrementally, one at a time. This is what we shall do – present a single input vector, calculate the resulting activations at the hidden and output layers.

First this single pass forward.

Activations

EASy

We need vectors(or arrays) to hold the activations at each layer.

```
float inputs[a+1];      inputs[a]=1.0; /* bias */
float hidden[b+1];      hidden[b]=1.0; /* bias */
float outputs[c];
float targets[c];       /* what the outputs should be */
```

Pick out one of the training set, and set the input values equal to this member.

Now calculate activations in hidden layer

Forward to Hidden layer

EASy

```
for (i=0;i<b;i++) {
        sum=0.0;
        for (j=0;j<a+1;j++)
                sum += inputs[j] * i_to_h_wts[j][i];
        hidden[i] = sigmoid(sum);
}
```

Using a sigmoid function you have written to calculate

$$y = g(x) = \frac{1}{1+e^{-x}}$$

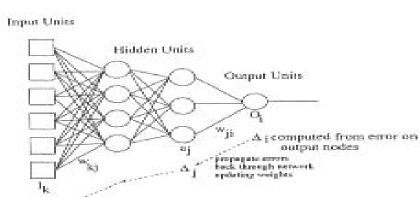
Forward to Output layer

EASy

```
for (i=0;i<c;i++) {
        sum=0.0;
        for (j=0;j<b+1;j++)
                sum += hidden[j] * h_to_o_wts[j][i];
        output[i] = sigmoid(sum);
}
```


End of forward pass

EASy

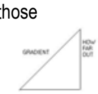


That has got us all the way forward.

Calculating delta's

EASy

Now for all the nodes, in all bar the first input layer, we are going to calculate deltas (appropriate basis for changes at those nodes).



float delta_hidden[b+1];

float delta_outputs[c];

The underlying formula used is $\partial_i = g'(x)\left[T_i - O_i\right]$

Which conveniently is $\partial_i = g(x)(1-g(x)\left[T_i - O_i\right]$

Deltas on output layer

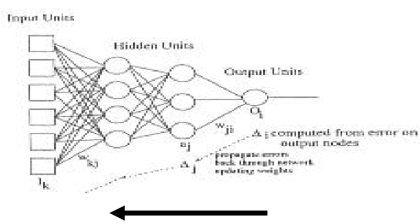
```
for (j=0;j<c;j++)
        delta_outputs[j] = outputs[j]*(1.0 – outputs[j]) *
                                        (target[j] – outputs[j]);
```

Store these deltas for the final output layer, and also use this to propagate the errors backward (using the weights on the connections) through to the hidden layer … …

Deltas on Hidden layer

```
for (j=0;j<b+1;j++) {
        error = 0.0;
        for (k=0;k<c;k++)
                error += h_to_o_wts[j][k] * delta_outputs[k];
        delta_hidden[j] = hidden[j] * (1.0 – hidden[j]) * error;
}
```


End of backward pass



That has got us all the way backward, calculating deltas.

Now the weight-changes

OK, we have calculated the errors at all the nodes, including hidden nodes. Let's use these to calculate weight changes everywhere – using a learning rate parameter eta γ

```
float delta_i_to_h_wts[a+1][b];
float delta_h_to_o_wts[b+1][c];
```


Calculate the weight-changes

```
for (j=0; j<c; j++)
        for (k=0; k<b+1; k++) {
                delta_h_to_o_wts[k][j] = eta *
                        delta_outputs[j] * hidden[k];
                h_to_o_wts[k][j] += delta_h_to_o_wts[k][j];
```

That gives new values for all the hidden-to-output weights

Calculate the weight-changes (2)

```
for (j=0; j<b; j++)
        for (k=0; k<a+1; k++) {
                delta_i_to_h_wts[k][j] = eta *
                        delta_hidden[j] * inputs[k];
                i_to_h_wts[k][j] += delta_i_to_h_wts[k][j];
```

That gives new values for all the inputs-to-hidden weights

How big is eta ?

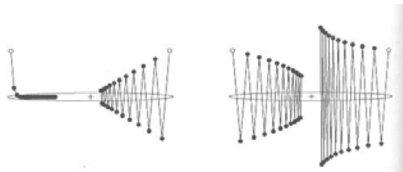
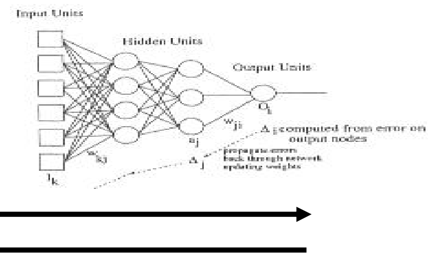


FIGURE 5.10 Gradient descent on a simple quadratic surface (the left and right parts are copies of the same surface). Four trajectories are shown, each for 20 steps from the open circle. The minimum is at the + and the ellipse shows a constant error contour. The only significant difference between the trajectories is the value of η, which was 0.02, 0.0476, 0.049, and 0.0505 from left to right.

For example, eta = 0.02 is a common rate to use.

End of forward and backward pass



Repeat many times

That was a single pass, based on a single member of the training set, and making small jiggles in the weights (based on the learning rate eta, e.g. γ = 1.0)

Repeat this lots of times for different members of the training set, indeed going back and using each member many times – each time making a small change in the weights.

Eventually (fingers crossed) the errors (Target – Output) will get satisfactorily small, and unless it has over-fitted the training set, the Black Box should generalise to unseen test data.

Wrapping up in a program

I presented the things-to-do in a pragmatic order, but for writing a program you have to wrap it up a bit differently.

Define all your weights, activations, deltas as arrays of floats (or doubles) first. Define your sigmoid function.

Write functions for a pass forward, and a pass backward.

Write a big loop that goes over many presentations of inputs.

All this is left as an exercise for the reader.

Problems ?

If there are, for instance, 100 weights to be jiggled around, then backprop is equivalent to gradient descent on a 100-dimensional error surface – like a marble rolling down towards the basin of minimum error.

(there are other methods, e.g. conjugate gradient descent, that might be faster).

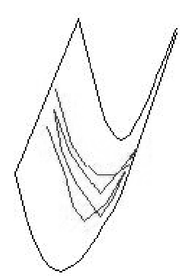
What about worries that 'the marble may get trapped in a local optimum'?

Actually, that rarely happens, though another problem is more frequent.

Valleys

Using the marble metaphor, there may well be valleys like this, with steep sides and a gently sloping floor.

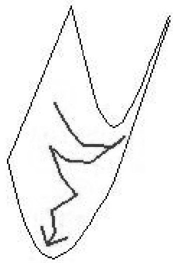
Gradient descent tends to waste time swooshing up and down each side of the valley (think marbles!)



Momentum

EASy

If you can add a **momentum** term, that tends to cancel out the back-and-forth movements and emphasise any consistent direction, then this will go down such valleys with gentle bottom-slopes much more successfully – faster.



Implementing momentum

EASy

This means keeping track of all the delta_weight values from the last pass, and making the new value of each delta_weight basically fairly similar to the previous value – I.e. give it momentum or 'reluctance to change'.

Look back a few slides to 'Calculate the weight changes' where I put a purple arrow →

Substitute

```
delta_wts[k][j] = eta * delta_outputs[j] * hidden[k] +
                  alpha * old_delta_wts[k][j];
```


Value for momentum

EASy

Alpha is the momentum factor, between 0 and 1. Typical value to use is 0.9.

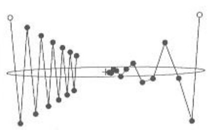


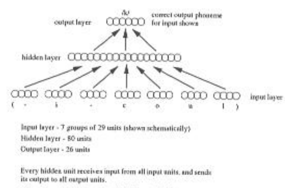
FIGURE 6.3 Gradient descent on the simple quadratic surface of Fig. 5.10. Both trajectories are for 12 steps with $\eta = 0.0476$, the best value in the absence of momentum. On the left there is no momentum ($\alpha = 0$), while $\alpha = 0.5$ on the right.

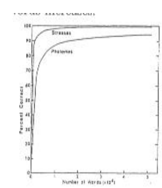
So common parameter setting rates (to be changed when appropriate) are: **Learning rate eta 0.02, momentum alpha 0.9**

Applications

EASy

You know have all the knowledge to write a back-prop program to tackle the most complex application – e.g. NETtalk (Sejnowski and Rosenberg 1986)





Exercise for the reader – Seminars week 4

EASy

Construct an ANN with 2 input nodes, 2 hidden nodes and one output node.

Use binary inputs – so there are 4 possible input vectors.

Train it on the XOR rule so that for these

Inputs	there are these Target outputs
00	0
01	1
10	1
11	0

Overfitting?

EASy

NB: with this particular problem, should you or should you not be worrying about overfitting?

See note in previous lecture on this.

Any Questions ?

EASy

Non-Symbolic AI lecture 9

Data-mining – using techniques such as Genetic Algorithms and Neural Networks.

Looking for statistically significant patterns hidden within loads of data – picking out the jewels from the rubbish.

Potentially valuable in searching for gold, searching for new drugs, searching for patterns in the stockmarket …

… etc etc etc.

Simple example

A survey of 20 CSAI graduates 5 years after graduating shows that 10 of them are dot.com millionaires and 11 of them are in jail.

There appears to be no particular correlation with their overall degree results … but perhaps there is a correlation with how well they did in one or more particular courses.

Maybe all the millionaires did well at Non-Symbolic AI **AND** badly at Intro to Logic Programming.

How can we find out if there are any such patterns?

The Outer loop

Split data into 50% train/ 50% test

Initialise at random a GA popn ofgenotypes

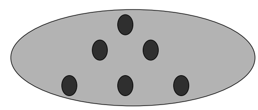
Run a GA for many generations, using mutation and recombination, to reduce errors on generalisation.

evaluate

Best trained ANN

Middle loop

Specification of 3 inputs (3 CSAI courses) to take notice of



Build 3-2-1 ANN

Initialise random wts

Train weights on train data

Calculate error on test data

Inner loop

Repeat over many presentations of training data

Present one set of inputs to the ANN

Pass forward thru layers to produce output

Error = difference between Output and Target

Back-propagate errors, and then adjust the weights

Inputs and Outputs

OK, lets take as inputs the scores on each of 30 courses taken over 3 years, each translated into a number between 0.0 and 1.0.

Lets take as the single output a number 1.0 for = millionaire, a number 0.0 for in jail.

So we are looking for a Black Box of some kind that will take 30 separate inputs, and output a single number that is its guess at how likely that person is to finish up a millionaire/in jail

Only 20 examples of data in our training set – is that a problem?

Overfitting ?

The small number of data points **could** be a problem.

If there is a clear and consistent simple rule that works well for the training set –

e.g. “All and only those who got over 60% on Non-Sym AI become millionaires, forget all the other courses as they are irrelevant”

-- then it might be reasonable to trust this.

But if the Black Box generates really complex rules involving scores on many courses, this may well be over-fitting the data.

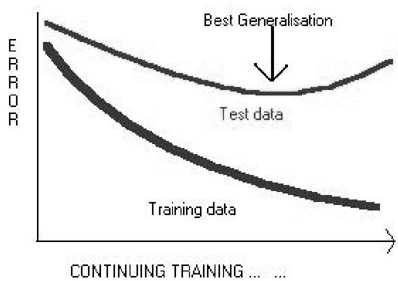
Training and Test data

The standard way to check for overfitting is to split the data into a training set and a test set.

Train the Black Box (… whatever method this may be …) on the training set, and then see how well it generalises to the unseen test data.

If there is a risk of overfitting, then the longer you train on the training data, the better the fit *on the training data* will be – but very likely at some stage the generalisation to test data will get worse.

Stop before you overfit



Calculating the errors

Errors, on either training or test data, depend on the difference between predictions (the actual output of the Black Box prediction machine, the ANN) and te known Target values.

Guessing 1.2 too high, or 1.2 too low usually counts as equally bad.

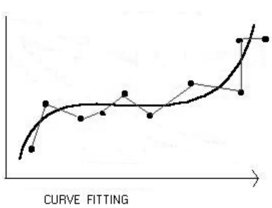
So the usual measure of errors is **Sum Squared Errors**

So over all the M members of a test set, the error measure will be

$$SSE = \sum_{i=1}^{i=M} \left(T_i - O_i\right)^2$$

Smoothing

Generally, smoother curves are more likely to generalise to new data than the (over-fitted) curves that go exactly through every training point.



Regularisation theory discusses topics such as how to get such smooth curves.

Regularisation

One method of ‘encouraging’ an ANN to produce smoother curves is to avoid lots of big positive or big negative weights – they are the ones that tend to produce jagged, non-smooth curves.

A good trick for doing this is **weight-decay**.

Introduce an extra routine into your program code, so that after every set of weight changes (through eg backprop) **all** the weights are decreased in (absolute) value by eg just 1%.

Then if the algorithm ‘has a choice’ between producing jagged curves or smooth ones, it will inevitably tend to opt for the smoother curves.

Too little data?

Back to the train/test approach for avoiding over-fitting.

That requires you to keep some part of the dataset back for testing, to check for generalisation.

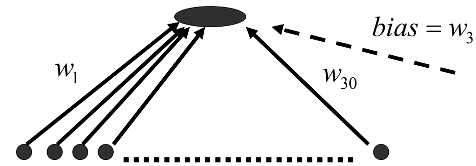
But if you only have 20 items of data anyway, dividing these into 2 parts makes the training data set even smaller.

This is often a serious problem.

Linear Black Boxes

If the problem is fairly simple, then there may be a simple linear formula that gives a good prediction.

$$out = w_1 In_1 + w_2 In_2 + \dots w_{30} In_{30}$$



Linear

So a single-layer perceptron could handle this.

Note that if you have 30 weights to discover (or 31 if you add a bias term), and only 20 data points, then there will be many different possible **exactly accurate** formulae (or Black Boxes).

$$eqn1 \dots 1.0 = 0.6w_1 + 0.3w_2 + \dots + 0.7w_{30} + w_{31}$$

$$\dots\dots\dots$$

$$\dots\dots\dots$$

$$eqn20 \dots 0.0 = 0.5w_1 + 0.55w_2 + \dots + 0.65w_{30} + w_{31}$$

20 simultaneous equations in 31 variables, to be solved.

Linear (ctd)

Basically, if you have **N**=31 parameters to find (weights + bias in a feedforward ANN, or formula as before), then

- If you have less than **N** equations (less than **N** examples in your training set) there are loads of possible 'exact' solutions (… many of which will not generalise …)
- If you have exactly **N** eqns there is 1 'exact' solution (and math techniques can find it, you don't need ANNs)
- If there are more than **N** eqns (datapoints – CSAI students) then probably there is not an exact linear solution.

Non-linear Black Boxes

A linear Black Box is something relatively simple like:

"Chance of becoming a millionaire = 2.0 * Non-Sym AI score minus 3.2 * Logic Prog score +0.5"

Otherwise rephrased as: $out = 2.0 In_{12} - 3.2 In_{17} + 0.5$

Non-linear means potentially much more complex formulae, like:-

$$y = 0.2x_1 - 3x_1^2 + 2x_1x_2 - x_3^{4.5} + \frac{x_6}{\sqrt{x_7}} - \log\left[2x_5 - \frac{x_4^3}{x_5}\right] + \dots$$

But don't worry …

… because a multi-layer perceptron or ANN – and normally adding just 1 hidden layer will do --- with non-linear transfer functions (the sigmoids) can do a pretty good job of making a Black Box that reproduces **any** such complex formula.

Provided that you have enough hidden nodes, and can find the right weights through a learning algorithm such as backprop.

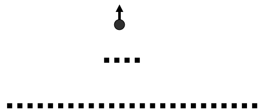
Still a few problems, though.

How many parameters to find ?

Suppose you now have 1output,

4 hidden,

and 30 inputs.



Counting biases as weights, you now have

(30 +1) * 4 + (4 + 1) * 1 = 129 weights or parameters to find

But you still only have 20 datapoints, 20 examples of CSAI students with known input ==> output results.

That simply isn't really enough. Suggestions ?

Suggestions

Well, one suggestion is to get a whole lot more data. Rather than 20 students from 1 year, get 1000 students from many years, then you will have enough to split into training and test sets, and train the ANN up nicely.

But this isn't always possible.

Maybe you have to hope that there is a simpler model, that only uses a much smaller number of the inputs.

For instance, maybe you can predict millionaire/in jail from results on only 3 CSAI courses rather than 30.

The Outer loop

Split data into 50% train/ 50% test

↓

Initialise at random a GA popn ofgenotypes

↓

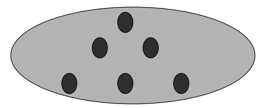
Run a GA for many generations, using mutation and recombination, to reduce errors on generalisation. → **evaluate** ←

↓

Best trained ANN

Middle loop

→ Specification of 3 inputs (3 CSAI courses) to take notice of



Build 3-2-1 ANN

Initialise random wts

Train weights on train data ⇄

← **Calculate error on test data**

Inner loop

→ **Repeat over many presentations of training data**

Present one set of inputs to the ANN

Pass forward thru layers to produce output

Error = difference between Output and Target

← **Back-propagate errors, and then adjust the weights**

Common data-mining problem

You have lots of *possibly* relevant input data, but it seems likely that most of it is irrelevant.

"90% of advertising is wasted – except we don't know *which* 90%"

Well, one suggestion is to try in turn **all** possible subsets of 10% of the inputs; each time generate a Black Box using just those, and see how well it generalises.

With the CSAI problem, try in turn **all** possible subsets of 3 courses out of 30.

Use a 3-input, 2-hidden, 1-output layer ANN.

Smaller ANN

EASy

This smaller ANN now has only (3+1)*2 +(2+1)*1 = 10 weights to learn, so we might have a chance even with only 20 or so training data.

We could split this into 10 training data, 10 test data.

Then for each selection of 3 courses (from 30):

- ❑Build a 3-2-1 ANN
- ❑Train on the training data
- ❑Test on the test data
- ❑The Error on the test data shows how good a generaliser it is

Compare all the possibilities

EASy

So now we will find that for many subsets-of-3 courses, we get lousy predictions of millionaire/in jail (for the previously unseen test data).

But hopefully some of them will generalise well – and we can pick out the best generaliser.

Then we will have two successes:

- ❑The most relevant 3 courses to consider (ignore the rest !)
- ❑And the non-linear combination of the scores on those 3 courses, that combine to give a good prediction.

A problem

EASy

There is at least one problem with all this method – anyone spot it?

We are going to have to do lots of choices of 3 subsets out of 30, and then do a whole neural network training routine each time.

How many of these will there be?

30*29*28/6 = 4,060 = rather a lot

Can we cut this down? -- Yes!

Use a Genetic Algorithm

EASy

There is a search space of thousands of subsets-of-3, and a ready-made **fitness function** for any one of them –

--- namely the Error on trying to generalise to test data.

(This is the kind of fitness-function one tries to make as **small** as possible).

So why not use a Genetic Algorithm to search through various possible sets-of-3 – and hopefully we can find the ideal set-of-3 with much less than exhaustive search of all the possibilities.

Genetic encoding

EASy

There are several different possible ways to genetically encode possible solutions.

A genotype must specify 3 out of the 30 possibilities. One simple way would be this:

Genotype[0] = 5 13 27

Genotype[1] = 11 17 29 etc etc

Then a mutation would change any selection to a new number in range 1-30

But problems …

Encoding problems

EASy

Firstly, a mutation might produce a genotype 5 13 13

Secondly, recombination between 2 parents might also produce 5 13 13

Where this is now a selection of 2 different, rather than 3 different numbers. Solutions?

One solution is to put special code in the program, so as to prohibit mutations or crossovers that produce such results.

Another solution

Another, easier, solution is just to ignore the problem!

If an input number is duplicated, then just carry on, and build the ANN with 2 inputs having he same vale. If the results are less fit (the trained ANN generalises badly to test data) then it will probably not have any offspring in the next generation of the GA.

GAs are forgiving

This is one example of how flexible and forgiving GAs are.

Often, rather than forcing the algorithm to generate and test only 'legal' possible solutions through using hard constraints, you can relax and allow all possibilities.

If the illegal ones don't work, then they will be eliminated naturally.

After all, that is what natural evolution is all about.

Another encoding

You could have a different genetic encoding, binary genotypes of length 30, where 0s mean 'ignore this factor (this CSAI course)' and 1s mean 'pick on this one'.

Then if genotypes have three 1s and 27 0s, they will effectively be selecting 3 out of 30.

Similar problems with mutations and recombination – but similar solutions to these problems can be found.

The Outer loop

Split data into 50% train/ 50% test

↓

Initialise at random a GA popn ofgenotypes

↓

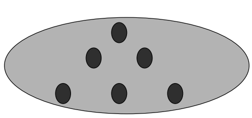
Run a GA for many generations, using mutation and recombination, to reduce errors on generalisation. ⇄ **evaluate**

↓

Best trained ANN

Middle loop

→ Specification of 3 inputs (3 CSAI courses) to take notice of



Build 3-2-1 ANN

Initialise random wts

Train weights on train data ⇄

← **Calculate error on test data**

Inner loop

→ **Repeat over many presentations of training data**

Present one set of inputs to the ANN

Pass forward thru layers to produce output

Error = difference between Output and Target

← **Back-propagate errors, and then adjust the weights**

Non-Symbolic AI lecture 10

EASy

Co-evolution – with animats in pursuit-evasion

… and in an application to 'sorting networks'

Coevolution of Pursuit and Evasion

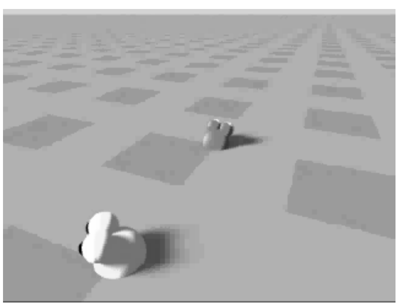
EASy

D. Cliff and G. F. Miller
``Co-Evolution of Pursuit and Evasion II: Simulation Methods and Results''. In
P. Maes, M. Mataric, J.-A. Meyer, J. Pollack, and S. W. Wilson (eds) From Animals to Animats 4
MIT Press Bradford Books, pp.506-515, 1996.

This paper, plus related ones, plus **mpegs** on
http://www.cogs.susx.ac.uk/users/davec/pe.html

Pursuit/Evasion – gen 200

EASy



Coevolution

EASy

Two (or more) species evolve in a situation where the selection pressure on one species (eg the Predators or Pursuers) depends (at least in part) on the current fitness of the other species (Prey or Evaders) .. .. and vice versa

Arm's Race, or 'Red Queen effect'
-- you run as fast as you can yet stay in same place
(... figuratively !)

This provides very much an **implicit** fitness function rather than an explicit one.

This study of coevolution

EASy

Studied in deliberately simplified environment - a 2-D infinite plane with no walls or obstacles, just one pursuer, one evader

Animats (animal/robot) - term often used in SAB

Motors:
These animats have left and right wheels. Variable forces can be applied L and R, simple Newtonian physics

Fuel use (from limited fuel tank) proportional to square of force. Friction acts to slow you down.

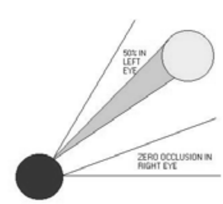
Sensors

EASy

Each animat has several (typically 2) simulated 'photoreceptors'. Position (relative to straght-ahead) and angle of acceptance (wide/narrow) is genetically specified -- and hence can co-evolve with the 'brain'

Each sensor returns proportion of of its angle-of-view which is **not** obscured by any object on horizon

Hence simulation is a very simplified version of real physics, but still has some significant element of physical plausibility.



Neural Network Control System

EASy

The control system is a CTRNN
(continuous-time recurrent neural network model),
of precisely the Beer type (see previous lecture).

Fully connected ANN, with (fixed) weights and biases that are genetically specified -- ie evolved.

2 neurons connected to 'eyes', 2 to motors.

Evolution

EASy

A Genotype for any one Animat specifies -
1) the sensory morphology
2) the architecture (weights etc) of the ANN

The Genetic Algorithm evolves 2 completely distinct populations ('species')

Spatially distributed GA -- individuals in the population are spread out over a 'mating' grid, and will only mate, and replace, close neighbours on this grid.

Evaluation

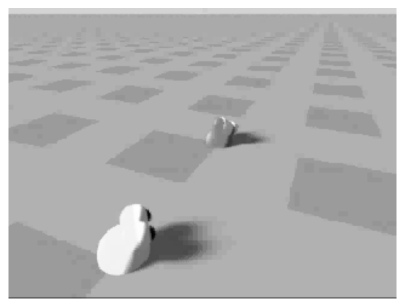
EASy

Evaluation:
All in the population of pursuers are tested against the same best-of-last-generation evader.
And vice versa.

Several trials from random starts:
Evader fitness = how long before caught
Pursuer fitness = ++ for 'approaching evader'
+ bonus for hit , sooner the bigger

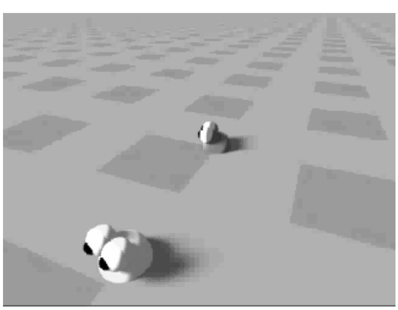
Random Start – gen 0

EASy



A Successful run – gen 999

EASy



Potential Circular Trap

EASy

That last picture showed successful pursuers/evaders from generation 999

But at gen 0, there was a pursuer which failed to catch an evader, and at gen 999 likewise.

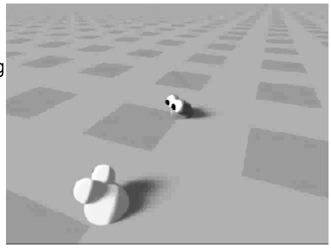
So in what sense has there been any 'advance'?

Possibility of 'no real advance' in coevolution
-- cf Stone Scissors Paper game, no strategy can be supreme for ever.

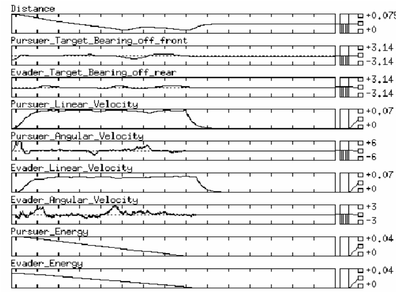
Possible Variations

Test evader from gen 200 against pursuer from gen 999.

Later work extended this idea, of monitoring current gen against best of previous gens. – Does this escape from the circular trap?



Analysis of behaviour



Applications

Can coevolution be used for engineering purposes?

Here is an example

Coevolving Parasites

"Coevolving Parasites improve Simulated Evolution as an Optimization Procedure". W. Daniel Hillis. In Artificial Life II, Langton Taylor Farmer Rasmussen (eds) Addison-Wesley (1991) pp 313-322

Danny Hillis -- Connection machines --powerful very distributed parallel machines.

This work done in late 1980s, 64,536 processors, populations 500 to 1000000, 'about 100 to 1000 generations per minute'

Evolving minimal *sorting networks*

Sorting Networks

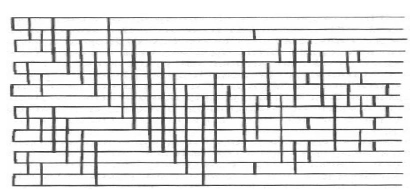
A sorting network is something that can be given **any** scrambled list of N objects with different values (here N=16) --- and it is in effect an algorithm that will systematically sort the list into order by a sequence of **'compare and maybe swap's.**

The sorting network is a series of pairs of numbers, [a b] which can be interpreted as:-

✓Compare the a^{th} and b^{th} items in your scrambled list.

✓ If in wrong order, swap, otherwise leave

Picturing Sorting Networks



Visual way to represent, 16 rows represent the 16 items to be re-ordered. Starting from left, the vertical bars show rows to be compared/swapped.Numbering rows from 0 to 15, above swaps are: [0 1] [2 3] ...[14 15] [0 2] [4 6] [8 10] ......

Minimal Sorting Networks

EASy

The previous diagram has a total of 60 swaps and was (in 1991) the shortest-known, discovered by MW Green

It is a **perfect** sorter, in that if you present it with **any** scrambled list, after going through all the 60 swaps from left to right then the list comes out perfectly ordered.

[note: for swaps shown as bars in same vertical column, it will not matter which is done first]

The problem is to find the shortest network, ideally better than this 60, which still sorts anything.

How to check if it works

EASy

Do you have to check if it sorts all possible combinations of numbers in the list – NO!

It can be shown that if a network correctly sorts any scrambled list of 0s and 1s (so that it finishes up with all the 1s at the top, all the 0s at the bottom), then the network will also sort any list of real-valued items.

So can test a 16-network exhaustively with only 2^{16} tests (about 32,000) – instead of 16 factorial (about 2×10^{13}).

But this is still a lot of tests -- can one save time? – YES!

Genetic Representation

EASy

We need a genetic encoding, so that strings of characters represent possible sorting networks.

But we are not sure how long any sorting network will be before we start – after all, we are looking for the shortest.

Hillis chose a sort-of-diploid encoding

haploid = 1 string
diploid = 2 strings

Diploid encoding

EASy

A codon pair looks like this or this:

```
          --------------               --------------
.... .... 0011 0101 ... ... ... ... ... ... ... ... 0011 0101 ...
.... .... 0011 1000 ... ... ... ... ... ... ... ... 0011 0101 ...
```

Where top and bottom are different, on left, this means test/swap [3 5] (binary 0011 and 0101), followed by test/swap [3 8] --- total 2 test/swaps

Where top and bottom are same, as on right, it is just test/swap [3 5] --- only one test/swap

Diploid encoding (ctd)

EASy

A full genotype is 60 such codon-pairs,

--- hence encoding between 60 and 120 test/swaps.

cf: homozygous / heterozygous (a bit different !)

Scoring

EASy

The population is initialised with everyone having the same first 32 exchanges (that are known to be sensible),
and thereafter randomised.

Then each network is tested on 'how well it sorts --
the percentage of input test scrambled lists which it sorts correctly.

Rather than testing on all 2^{16} test cases, it could be tested on a random sample.

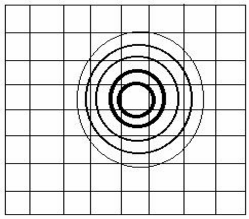
OR (see later) the test cases could be chosen cleverly – coevolution.

Spatially Distributed GA

Tournaments:

pick pairs of contestants in local neighbourhood

(Gaussian spread, nearer is more likely)



Reproduction

Tournament: from pair of contestants, compare scores, winner over-writes loser (ie then has 2 copies).

Mating: then select mates locally, with same principles

Recombination to produce offspring
(Hillis actually had 15 crossover points '1 per chromosome')

Mutation: one bit-flip per 1000 sites.

Results – without coevolution

Typical run like this -- without coevolution -- for up to 5000 generations, with a popn of 64536.

Best scores = sorting networks of 65 exchanges
-- target was 60.

How can one improve this through coevolution ?

Inefficiencies - 1

Two main sources of inefficiency in the GA without coevolution:

(1) Local optima -- once the population had found a 3/4 decent solution, quite probably all the neighbouring solutions (genetically similar) were less fit -- so the population would have to cross a valley to reach 'higher ground'.

Inefficiencies - 2

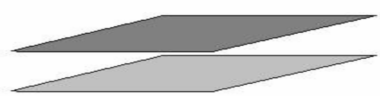
(2) Inefficiency in testing -- once popn was ¾ decent, they all passed most of the test cases, so little differences in scores.

The answer: co-evolve a separate population of parasite test-cases, which themselves have a fitness function designed to make them as hard as possible for the sorting networks.

This solves both inefficiencies (1) and (2).

Parasite coevolution can generate genetic diversity
(cf. W Hamilton)

2 populations – sorters and parasites



The population of sorting networks is already spatially distributed on one grid. Have a population of parasites likewise distributed on a similar grid, overlaid.

Each parasite is a genetically specified group of 10 to 20 test cases -- rather than all the 2^{16} possible ones.

Scoring each population

Each sorting network is tested against the parasite that is on corresponding grid square. The score of the sorting network is 'what proportion of tests does it pass'

The score of the parasite is 'how many tests does it fail the sorter on'

Networks get selected, mated, and reproduce on their grid, parasites completely separately on theirs.

Results improved to a minimum size of 61
(has it been beaten since?)

Benefits of Coevolution

Prevents getting stuck in local optima -- as soon as this happens, the parasites evolve to zap them.

Population is in a constant state of flux.

Second advantage: testing is more efficient --
need only test on a few difficult test cases, which themselves change appropriately according to circumstances.

Hence computationally more efficient.

Non-Symbolic AI lecture 11

Evolution of Communication

In particular, 2 papers from Proc of Artificial Life II
ed. CG Langton C Taylor JD Farmer and S Rasmussen
Addison Wesley 1991

(1) Bruce MacLennan
Synthetic Ethology:
An Approach to the Study of Communication (pp 631-658)
THIS LECTURE

Another for reference

and (2) Greg Werner and Michael Dyer
Evolution of Communication in Artificial Organisms
(pp 659 - 687)

There are many more recent papers on all aspects of communication, in fact one of the more popular Alife subject areas. Not all the work is good!

Other work

Couple of other mentions of recent stuff:

Luc Steels 'Talking Heads'

Ezequiel di Paolo, on 'Social Coordination',
DPhil thesis plus papers via web page
http://www.cogs.susx.ac.uk/users/ezequiel/

What is communication ?

What is communication, what is meaning? Cannot divorce these questions from philosophical issues. Here is a very partial survey:

Naive and discredited **denotational** theory of meaning
'the meaning of a word is the thing that it denotes'

bit like a luggage-label.

Runs into problems, what does 'of' and 'the' denote?

What is it -- ctd

Then along came sensible people like Wittgenstein -- the idea of a 'language game'.

"Howzaaat?" makes sense in the context of a game of cricket.

The meaning of language is grounded in its use in
a social context. The same words mean different things
in different contexts.

Social context

cf Heidegger -- our use of language is part of our culturally constituted and situated world of needs,concerns and skilful behaviour.

SO... you cannot study language separately from some social world in which it makes sense.

Synthetic Ethology

So, (says MacLennan) we must set up some simulated world, some ethology in which to study language.

Ethology = looking at behaviour of organisms *within* their environment (not a Skinner box)

Burghardt's definition

GM Burghardt (see refs in MacLennan)
Definition of communication (see any problems with it?):

"Communication is the phenomenon of one organism producing a signal that, when responded to by another organism, confers some advantage (or the statistical probability of it) to the signaler or its group"

Grounding in evolutionary advantage

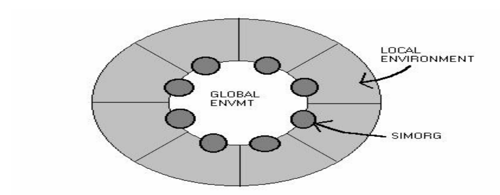
Criticisms

Ezequiel Di Paolo's methodological criticism of Burghardt:

"This mixes up a characterisation of the phenomenon of communication with an (admittedly plausible) explanation of how it arose"

Another dodgy area: treatment of *'communication'* as '*transmission of information*' without being rigorous about definition of **information.**

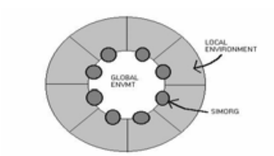
Simorgs



Simulated organisms: why should there be any *need* to communicate (MacLennan asks..) ??

Simorg world

OK, set it up so that each simorg has a *private* world, a **local environment** which only they can 'see', With one of 8 possible symbols *a b c d e f g h*



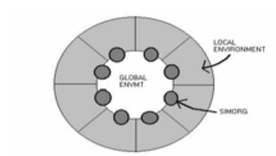
Plus there is a *shared* public world, a **global environment** in which any simorg can make or sense a symbol.
-- one of 8 possible symbols *p q r s t u v w*

Why communicate ?

Simorgs have to:

(a) 'try to communicate their private symbol' *and*

(b) 'try to guess the previous guy's'



Each simorg can **write** a symbol *p-to-w* in global env ('emit') and **raise a flag** with symbol *a-to-h* ('act')

Writing a new symbol over-writes the old.

Simorg actions

EASy

When it is its turn, a simorg both writes a symbol and raises a flag, eg [*q, d*] -- depending on what its genotype 'tells it to do' (see later for explanation).

What counts as success is when it raises a flag *matching* the **private symbol** of the simorg who had the previous turn (normally turns go round clockwise)

Ie if simorg 5 does [*q, d*], when simorg 4's private symbol happened to be *d*', then this counts as 'successful communication' (via the global symbols) and **both** simorg4 and simorg5 get a point !

Evaluating their success

EASy

How do you test them all, give them scores? --

(A) minor cycle -- all private symbols are set arbitrarily by 'God', turns travel 10 times round the ring, tot up scores

(B) major cycle -- do 5 minor cycles, re-randomising all the private symbols before each major cycle.

Total score from (B) for each simorg is their 'fitness'

Simorg genotype

EASy

Each simorg faces 64 possible different situations --
8 symbols *a-to-h* privately, plus
8 symbols *p-to-w* in the public global space.

For each of these 64 possibilities, it has a genetically specified pair of outputs such as [*q, d*] which means '**write** *q* in public space, **raise flag** *d*'

So a genotype is 64 such pairs, eg

[*q d*] [*w f*] [*v c*]... ... 64 pairs long .. .. [*r a*]

The Evolutionary Algorithm

EASy

A Genetic Algorithm selects parents according to fitness (actually he used a particular form of steady-state GA) and offspring generated by crossover and mutation, treating pairs [*q d*] as a single gene.

NOTE: the importance of using steady-state GA, where only one simorg dies and is replaced at a time -- it allows for '*cultural transmission*', since the new simorg is born into '*an existing community*'

Adding learning

EASy

To complicate matters, in some experiments there was an additional factor he calls 'learning'.

Think of the genotype as DNA, which is inherited as characters.

When a simorg is born, it translates its DNA into a lookup table, or transition table, which is used to determine its actions.

How 'learning' works

EASy

WHEN learning is enabled, then after each action it is checked to see if it 'raised the wrong flag'.

If so, the entry in the lookup table is changed so that another time it would 'raise the correct flag' (ie matching previous simorg's private symbol)

BUT this change is *only* made to the phenotype, affecting scores and fitness, **NO CHANGE** is made to the genotype (which is what will be passed on to offspring) -- ie it is *not* Lamarckian.

How to interpret results?

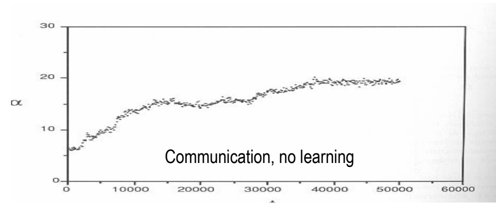
Suppose you run an experiment, with 100 simorgs in a ring, 8 private (*a-h*) and 8 public (*p-w*) symbols, for 5000 new births.

You *may* find communication taking place, after selection for increased fitness, with some (initially arbitrary) code being used such as
'if my private symbol is *a*, write a *p* into public space -- if you see a *p*, raise a flag with *a*' -- etc etc.

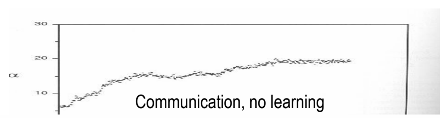
But how can you objectively check whether there really is some communication?

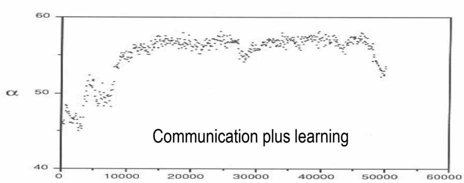
Tests for 'communication'

(1) Compare results doing as above with results when the global envt symbol is vandalised at every opportunity -- ie replaced with a random symbol. Fitnesses should differ when there is/is not such vandalism.



Comparison with learning





Dialects test

(2) Second way to test for communication: keep a record of every symbol/situation pair, such as

'see a global p, raise flag a' -- how often seen?
'see a global p, raise flag b' -- ditto
... ...
see a global w, raise flag h' -- ditto

If no communication, one should not expect any particular pattern to emerge, whereas with communication you should expect such statistics to have some discernible structure.

Evidence of dialects

TABLE 3 Denotation Matrix, Communication Permitted and Learning Disabled

	situation							
symbol	0	1	2	3	4	5	6	7
0	695	5749	0	1157	0	2054	101	0
1	4242	11	1702	0	0	0	1	0
2	855	0	0	0	0	603	862	20
3	0	0	0	0	1003	430	0	1091
4	0	0	0	0	0	0	2756	464
5	0	0	40	0	548	0	817	0
6	1089	90	1	281	346	268	0	62
7	0	201	0	288	0	0	2	0

$V = 2.272352$
$H = 3.915812$
$\eta = 0.3052707$

Comments

- Rarely a one-to-one denotation in the matrix
- Not always symmetric
- Probabilistic -- symbol 4 'means' situation 6 84% of time, means situation 7 16% of time.

Interesting comment: this method of GA saw communication arising, ---- **but** the original experiments were deterministic in the sense that: "least fit always died, the two fittest simorgs always bred to produce the replacement offspring" -- in these original experiments communication never arose !

Some different views

EASy

See Ezequiel di Paolo
"An investigation into the evolution of communication"
Adaptive Behavior, vol 6 no 2, pp 285-324 (1998)
via his web page
http://www.cogs.susx.ac.uk/users/ezequiel/

Suggests the idea of information as a commodity has contaminated many peoples' views, including MacLennan.

MacLennan explicitly sets up the scenario such that some information is not available to everyone.

Communication without such information

EASy

BUT there are often phenomena that we think of as communication when all relevant info is available to all concerned
-- eg within a wolf pack forming a coordinated pattern for hunting prey.

Signals as actions rather than packages of information.

Communication as Social coordinated activity.

Autopoiesis

EASy

Maturana and Varela 1988
The Tree of Knowledge: the biological roots of human understanding. Shambala Press, Boston

If 2 or more organisms have their activities coupled
(in a dynamical systems sense -- each perturbs the activity of the other)
then their activities become coordinated.

This establishes a consensual domain.

Communication as interaction in a consensual domain

EASy

Communication can be defined as the behavioural coordination that we can observe as a result of the interactions that occur in a consensual domain.

This is complex stuff, worth pursuing. Di Paolo's work offers a good way in.

Non Symbolic AI Lecture 12

Some more on Evolutionary Algorithms

1) Genetic Programming – GP
2) Species Adaptation Genetic Algorithms -- SAGA

Genetic Programming

GP (a play on General Purpose) is a development of GAs where the genotypes are pretty explicitly pieces of computer code.

This has been widely promoted by John Koza, in a series of books and videos, eg:

Genetic Programming, John Koza, MIT Press 1992
Followed by volumes II and III

Sort of: using NSAI techniques to evolve programs (of symbols!)

GP - Problems

At first sight there are a number of problems with evolving program code, including

(Problem 1) How can you avoid genetic operators such as recombination and mutation completely screwing up any half-decent programs that might have evolved?

(Problem 2) How can you evaluate a possible program, give it a fitness score?

GP solution to problem 1

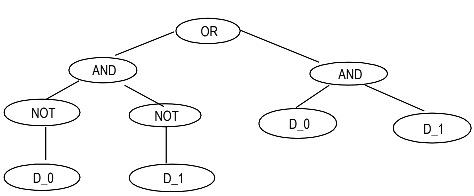
Both these 2 problems were basically cracked in work that predated Koza:

N. Cramer *A representation for the adaptive generation of simple sequential programs* In J Grefenstette (ed) *Proc of Int Conf on Genetic Algorithms*, Lawrence Erlbaum, 1985.

(1) Use LISP-like programs, which can be easily pictured as rooted point-labelled trees with ordered branches.

Picturing a Lisp program

(OR (AND (NOT D_0) (NOT D_1)) (AND D_0 D_1)



Recombining 2 Lisp programs

Picture each of 2 parent Lisp programs in tree form.

Then **recombination** between 2 parents involves taking their 2 trees, choosing random points to chop off sub-trees, and swapping the sub-trees.

This maintains the general form of a program, whilst swapping around the component parts of programs.

Mutating a Lisp program

EASy

Mutation in GP is rarely used:

"Nonetheless, it is important to recognize that the mutation operator is a relatively unimportant secondary operation in the conventional GA", says Koza (op. cit. p. 105), citing Holland 1975 and Goldberg 1989.

When it is used, it operates by randomly choosing a subtree, and replacing it with a randomly generated new subtree.

GP solution to problem 2

EASy

How do you measure the fitness of a program? (this was the 2nd problem mentioned earlier)

Usually, fitness is measured by considering a fixed number of test-cases where the correct performance of the program is known.

Put a number on the error produced for each test-case, sum up the (absolute) value of these errors => fitness.

Normalising fitness

EASy

Often there is some adjustment or normalisation, eg so that normalised fitness nf ranges within bounds 0 < nf < 1, increases for better individuals, and sum(nf)=1.

Normalised fitness => fitness-proportionate selection

A typical GP run

EASy

- has a large population, size 500 up to 640,000
- runs for 51 generations (random initial + 50 more)
- has 90% crossover -- ie from a population of 500, 450 individuals (225 pairs) are selected (weighted towards fitter) to be parents
- and 10% selected for straight reproduction (copying)
- no mutation
- maximum limit on depths of new trees created
- selection is fitness-proportionate

Different problems tackled by GP

EASy

Koza's GP books detail applications of GP to an enormous variety of problems.

- Eg finding formulas to fit data
- control problems (eg PacMan)
- evolution of subsumption architectures
- evolution of building blocks (ADF automatic definition of (sub-)functions)
- etc etc etc

Choice of primitives

EASy

In each case the primitives, the basic symbols available to go into the programs, must be carefully chosen to be appropriate to the problem.

Eg for PacMan:
Advance-to-Pill
Retreat-from-Pill
Distance-to-Pill
... ...
IFB(C D) If monsters blue, do C, otherwise D
IFLTE(A B C D) If A<=B, do C, otherwise do D

Criticism of GP

GP has been used with some success in a wide range of domains. There are some criticisms:

Much of the work is in choice of primitives

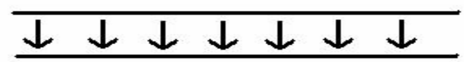
Successes have not been in General Purpose ('GP') programming -- very limited success with partial recursive programs (eg those with a DO_UNTIL command)

But different Evolutionary Algorithm ADATE is far better than GP on this.

Wide and short, not long and thin

GP practitioners such as Koza typically use very large populations such as 640,000 for only 51 generations.

This is closer to random search than to evolution.
Very WIDE and SHORT



SAGA

SAGA, for Species Adaptation Genetic Algorithms, is in many respects the very opposite of GP. Papers on my web page
http://www.informatics.susx.ac.uk/users/inmanh/

It is intended for *very long term evolution*, for design problems which inevitably take many many generations, quite possibly through incremental stepping-stones.

Long and thin, not wide and short

So in contrast to GP, there is typically a relatively small population (30-100) for many generations (eg 1000s or 10,000s).

'*Long and narrow*' not 'wide and shallow'

The population is very largely 'genetically converged' -- ie all members genetically very similar, like a plant or animal ***SPECIES.***

Mutation vs. Recombination

Mutation is the main genetic operator, adding diversity and change to the population.

Recombination is only secondary (though useful).

This is completely contrary to the usual emphasis in GP

Convergence (1)

The term 'convergence' is often used in a confused way in the GA/GP literature. People fail to realise that it can be applied with at least two different meanings.

(1) Genetic convergence -- when the genetic 'spread' of the population has settled down to its 'normal' value, which is some balance between:

- selection of the fittest -> reduces spread, and
- mutation -> increases spread.

Convergence (2)

EASy

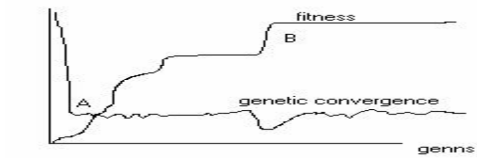
(2) convergence of the fitness of the population onto its final value, or (very similarly) convergence of the 'search' of the popn onto its final resting-place.

In sense (2) people talk of 'premature convergence', particularly when they are worried about the population converging onto a local optimum in the fitness landscape, one that is very different from the global optimum.

A common, completely false, myth is that convergence in sense (1) implies convergence in sense (2).

Monitoring Genetic Convergence

EASy



B is the point of convergence (defn 2), often after 'punctuated equilibria'

A is the point of genetic convergence (defn 1), which may well be (surprisingly?) within the first 10 or so generations !

Long term incremental evolution

EASy

SAGA was originally developed with a view to long term incremental evolution, where one would start with (relatively) short genotypes encoding (eg) relatively simple robot control systems ...

... then over time evolution would move to longer genotypes for more complex control systems.

In this long term evolution it is clear that the population will be genetically converged for all bar the very start

… also long term non-incremental evolution

EASy

BUT though SAGA was originally developed with the view of long term incremental evolution (where genotype lengths probably increased from short, originally, to long and then even longer…)

It soon became apparent that the lessons of evolution-with-a-genetically-converged species were **ALSO** applicable to *any* long term evolution, even if genotype lengths remain the same!

Consequences of convergence (1)

EASy

It soon became apparent that even with fixed-length genotypes, one still has genetic convergence of the population from virtually the start -- even though this is not widely recognised.

Consequences: recombination does not 'mix-and-match' building blocks quite as expected -- because typically the bits swapped from mum are very similar to the equivalent from dad.

Consequences of convergence (2)

EASy

Evolution does not stop with a genetically-converged population (GCP) -- eg the human species, and our ancestors, have evolved as a GCP (… or *species)* for 4 billion years, with incredible changes.

Mutation is the main engine of evolutionary change, and should be set at an appropriate rate – which to a first approximation (depending on selection) is:

… …

Junk DNA

A high proportion – indeed most – of human and other animal/plant DNA appears to be 'junk'.

I.e., not used, not 'translated' or 'interpreted'. So what is it doing there?

"Rubbish is what you chuck out, junk is what you put in the attic in case you might need it later"

Some genes might be unnecessarily duplicated by accident – and later the spare copies mutated into something else useful.

Optimal mutation rate – for *Binary-type* genotypes

One proposal (with some small print):

1 mutation in the expressed (non-junk) part of the genotype

CF phages with 4500 'characters' in genotype
humans with 3,000,000,000 'characters'

-- both have something of the order of 1 mutation per genotype (or at least per non-junk part)

Why should there be an optimal mut-rate?

Mutation rate too low, in limit zero, would mean no further change, evolution ceases -> no good

Mutation rate too high, eg every bit flipped at random, implies random search -> no good.

What decides the ideal rate?

Some ideal rate (or range of rates) inbetween -- but where ?

Very rough version of argument: to a first approximation (tho not a 2nd !) at any stage in evolution some N bits of the genotype are crucial -- any mutations there are probably fatal -- while the rest is junk.

The *error threshold* shows that maximum mutation rate survivable under these circumstances is of the order of 1 mutation in the N bits

So what is SAGA?

Species Adaptation Genetic Algorithms

SAGA is not a specific GA, it is just a set of guidelines for setting the parameters of your GA when used on any long term evolutionary problem -- with or without change in genotype length

1) Expect the population to genetically converge within the very first few generations

2) Mutation is the important genetic operator

… SAGA ctd

3) Set the mutation rate to something of the order of 1 mutation per genotype. Subject to small print below.
4) Small print 1: it is the mutations in the non-junk part that matter – so if only 1/3 is non-junk, you will need 3 mutns per genotype,
5) Small print 2: this is only for standard selection pressure (as in tournaments of Microbial GA). Stonger selection needs more mutation. If abnormal selection pressures, for 1 substitute log(S) where S is the expected no. of offspring of the fittest member

… SAGA ctd …

Often there is quite a safe range of mutation rates around this value -- ie although it is important to be in the right ballpark, exact value not too critical

Recombination generally assists evolution a bit

Expect fitness to carry on increasing for many many generations

Mutation rates – Summing up

IMPORTANT: very different rules for binary and for real-valued genotypes.

BINARY or similar: Here a mutation flips from 0 to 1, or from 1 to 0.

Mutation rate – use SAGA principle of 1 mutation per genotype, adjusted if necessary for junk DNA and for abnormal selection pressure.

This may not be the best rate – but at least a good starting place.

Mutation Rates – for real-valued genotypes

BUT the rule for real-valued genotypes is different.

Real-values such as when a genotype encodes the weights of an Artificial Neural Network.

Mutating a real number (float or double) – you are probably best off changing it by a small “creep factor” (cf Lec 3).

And then it may well be a good idea to change *every* locus on the genotype by a small creep factor – as compared to changing just *one* locus in a binary genotype by a big change (0 / 1)

Lots of little changes

In, e.g. an Artificial Neural Network, a learning algorithm typically jiggles *every* weight by a little bit.

It is much the same when, in a GA with real-values on the genotype, a mutation jiggles (or ‘creeps’) *every* locus, every real value, by a little bit.

Selection pressures

It is very tempting to use a high selection pressure for Genetic Algorithms/Evolutionary Algorithms --

~~“Just select the very fittest as a parent and throw away the rest”~~

This is almost always a bad idea – evolutionary search gets stuck in a dead end.

I recommend standard selection – as in Microbial GA, pick winner of tournament of size 2 – and then adjust the mutation rate appropriately. But you can experiment.

The End

Non Symbolic AI - Lecture 13

Symbolic AI is often associated with the idea that "*Intelligence is Computation*" and *"The Brain is a Computer"*

Non Symbolic AI is often associated with the idea that *"Intelligence is Adaptive Behaviour"* and *"it arises from the dynamical interactions of networks of simple components"*

Eg **ANNs** (Artificial Neural Networks)

Cellular Automata (and Random Boolean Networks)

These are another class of systems that fit this last description.

In the wide spectrum of approaches to synthesising 'lifelike' behaviour, **CA**s and **RBN**s are amongst the most abstract and mathematical.

A lot of the interest in this comes from people with a Physics background. Cf. Los Alamos, Santa Fe, the 'chaos cabal' .

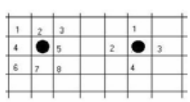
(pop book on the chaos cabal: "The Newtonian Casino" T. Bass 1990 Longmans, (US= "The Eudaemonic Pie" 1985)

The Game of Life

Best known CA is John Horton Conway's "Game of Life". Invented 1970 in Cambridge.

Objective: To make a 'game' as unpredictable as possible with the simplest possible rules.

2-dimensional grid of squares on a (possibly infinite) plane. Each square can be blank (white) or occupied (black).



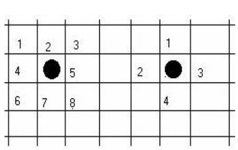
More Game of Life

At any time there are a number of squares with black dots. At the 'regular tick of a clock' all squares are updated simultaneously, according to a few simple rules, depending on the **local situation**.

For the 'Game of Life' **local situation** means, for any one cell, the current values of itself and 8 immediate neighbours ('Moore neighbourhood')

Neighbourhoods

8 immediate neighbours = 'Moore neighbourhood', on L
For different CAs, different neighbourhoods might be chosen; e.g. the 'von Neumann neighbourhood', on R.



Readable pop sci on CAs: William Poundstone "The Recursive Universe" OUP 1985

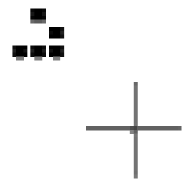
Game of Life: rules

Update rule for each cell:

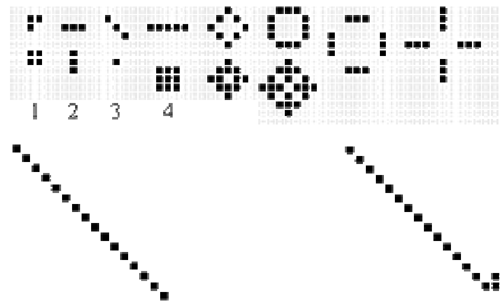
✓**If** you have exactly 2 'on' nbrs (ie 2 blacks) stay the same
✓**If** you have exactly 3 'on' nbrs you will be 'on' (black) next timestep (ie change to on if you are blank, and remain on if you already are)
✓**If** you have less than 2, or more than 3 on nbrs you will be off (blank) next timestep

Thats all !

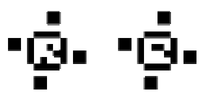
Glider



Sequences



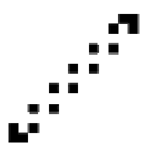
More



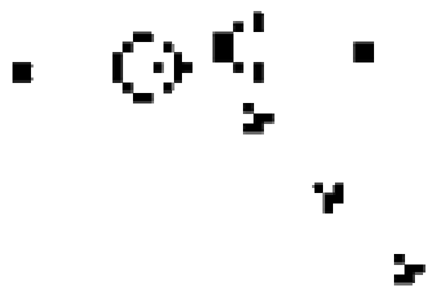
Sequence leading to Blinkers

Clock

Barber's pole



A Glider Gun



More formal definition of CA

✓A regular lattice	eg. grid
✓of finite automata	eg. cells
✓each of which can be in one of a finite number of states	eg. black/white tho could be 10 or 100
✓transitions between states are governed by a state-transition table	eg. GoL rules

✓input to rule-table = state of cell and specified local neighbourhood (In GoL 2 ^9 = 512 inputs)
✓output of rule-table = next state of that cell

CAs

All automata in the lattice (all cells on the grid) obey the same transition table, and are updated simultaneously.

From any starting setup on the lattice, at each timestep everything changes **deterministically** according to the rule-table.

Game of Life: rules (again!)

Update rule for each cell:

✓**If** you have exactly 2 'on' nbrs (ie 2 blacks) stay the same
✓**If** you have exactly 3 'on' nbrs you will be 'on' (black) next timestep (ie change to on if you are blank, and remain on if you already are)
✓**If** you have less than 2, or more than 3 on nbrs you will be off (blank) next timestep

Thats all !

Alternative version of rules

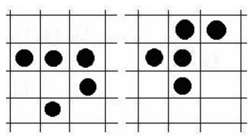
Every cell is updated simultaneously, according to these rules, at each timestep.

Alternative (equivalent) formulation of Game of Life rules:

0,1 nbrs = starve, die	2 nbrs = stay alive
3 nbrs = new birth	4+ nbrs = stifle, die

Gliders and pentominoes

On the left: a **'Glider'**
On a clear background, this shape will 'move' to the NorthEast one cell diagonally after 4 timesteps.



Each cell does not 'move', but the 'pattern of cells‘ can be seen by an observer as a glider travelling across the background.

Emergence

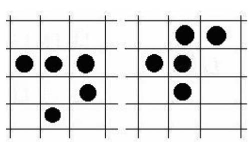
This behaviour can be observed as ‘the movement of a glider’, even though no glider was mentioned in the rules.

'Emergent' behaviour at a higher level of description, emerging from simple low-level rules.

Emergence = emergence-in-the-eye-of-the-beholder (dangerous word, controversial)

Pentominoes

On the right: a 'pentomino'. Simple starting state on a blank background => immense complexity, over 1000 steps before it settles.



(Pop Sci) William Poundstone "The Recursive Universe" OUP 1985

(Primordial Soup kitchen) http://psoup.math.wisc.edu/kitchen.html
http://www.math.com/students/wonders/life/life.html
http://www.bitstorm.org/gameoflife/

Game of Life - implications

Typical Artificial Life, or Non-Symbolic AI, computational paradigm:
✓bottom-up
✓parallel
✓locally-determined

Complex behaviour from (... emergent from ...) simple rules.

Gliders, blocks, traffic lights, blinkers, glider-guns, eaters, puffer-trains ...

Game of Life as a Computer ?

Higher-level units in GoL can in principle be assembled into complex 'machines' -- even into a full computer, or Universal Turing Machine.

(Berlekamp, Conway and Guy, "Winning Ways" vol 2, Academic Press New York 1982)

'Computer memory' held as 'bits' denoted by 'blocks' laid out in a row stretching out as a potentially infinite 'tape'. Bits can be turned on/off by well-aimed gliders.

Self-reproducing CAs

von Neumann saw CAs as a good framework for studying the necessary and sufficient conditions for
self-replication of structures.

von N's approach: self-rep of abstract structures, in the sense that gliders are abstract structures.

His CA had 29 possible states for each cell (compare with Game of Life 2, black and white) and his minimum self-rep structure had some 200,000 cells.

Self-rep and DNA

This was early 1950s, pre-discovery of DNA, but von N's machine had clear analogue of DNA which is **both:**

✓used to determine pattern of 'body'
interpreted

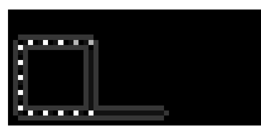
✓and itself copied directly
copied without interpretation as a symbol string

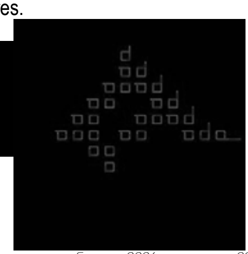
Simplest general logical form of reproduction (?)

How simple can you get?

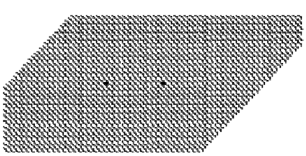
Langton's Loops

Chris Langton formulated a much simpler form of self-rep structure - Langton's loops - with only a few different states, and only small starting structures.





Snowflakes



One dimensional CAs

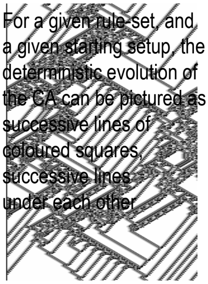
Game of Life is 2-D. Many simpler 1-D CAs have been studied, indeed whole classes of CAs have been.

Eg. a 1-D CA with 5 states (a b c d and - = blank) can have current state of lattice such as

- - - a - b c d c a - - -

or pictorially with coloured squares instead of a b c d
Then neighbours of each cell are (typically) one on each side, or 2 on each side, or ... ...

Spacetime picture



For a given rule-set, and a given starting setup, the deterministic evolution of the CA can be pictured as successive lines of coloured squares, successive lines under each other

DDLab

That coloured spacetime picture was taken from Andy Wuensche's page
www.ddlab.com

DDLab is

Discrete Dynamics Lab

Wuensche's work allows one to run CAs backwards, to see what previous state(s) of the world could (according to the rules) have preceded the present state.

Wolfram's CA classes 1,2

From observation, initially of 1-D CA spacetime patterns, Wolfram noticed 4 different classes of rule-sets. Any particular rule-set falls into one of these:-:

CLASS 1: From any starting setup, pattern converges to all blank -- **fixed attractor**

CLASS 2: From any start, goes to a limit cycle, repeats same sequence of patterns for ever. **-- cyclic attractors**

Wolfram's CA classes 3,4

CLASS 3: From any start, patterns emerge and continue continue without repetition for a very long time (could only be 'forever' in infinite grid)

CLASS 4: Turbulent mess, no patterns to be seen.

Classes 1 and 2 are boring, Class 4 is messy,
Class 3 is **'At the Edge of Chaos'** - at the transition between order and chaos -- where Game of Life is!.

Applications of CAs

Modelling physical phenomena, eg diffusion.

Image processing, eg blurring, aliasing, deblurring.

Danny Hillis's Connection Machine based on CAs.

Modelling competition of plants or organisms within some space or environment.

To Finish up -- Hot Research topics

Hot research topics in NSAI and Alife – eg looking ahead to 3rd year u/g project topics

With personal bias and prejudice

- Evolutionary Robotics
- Hardware Evolution
- Molecular Drug Design
- Incremental Artificial Evolution (SAGA)
- Neutral Networks (Neutral with a T)

That last topic …

Quick few slides on Neutral Networks

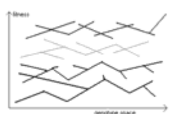
Neutral Networks

Neutral Networks have been much discussed in the context of RNA (including in Alife circles) but they are overdue for appreciation as crucially relevant to all of evolutionary computation

-- and particularly much Alife artificial evolution, just because that is macro-evolution

How do Neutral Nets work ?

If local optima are not points to get stuck on, but ridges of the same fitness percolating through genotype space, then you can escape.



Conditions you need are:

- Very high dimensional search space
- Very many->one genotype->phenotype mapping
- Some statistical correlation structure to the search space

What happens ?

Punctuated equilibria

Most of evolution is running around the current neutral network ‘looking for’ a portal to a higher one.

Much of macro-evolution, the sort that is relevant to Alife as well as RNA worlds, just is this kind of evolution – the LONG HAUL or SAGA, rather than the BIG BANG.

Recent Work

… relevant outside the RNA world includes theoretical work by Erik van Nimwegen and colleagues at SFI.

Lionel Barnett at Sussex - “Ruggedness and Neutrality: the NKp family of Fitness Landscapes” – Alife VI

Rob Shipman and colleagues from BT at this Alife VII

But do the conditions for NNs to exist actually apply in real practical problems? -- recent work suggests probably YES!

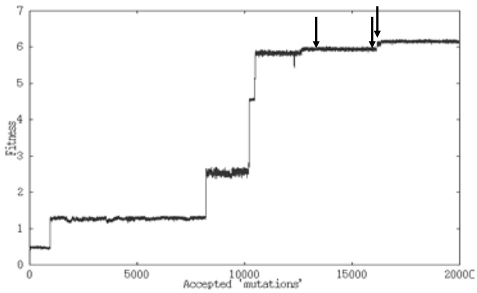
Evolvable Hardware

Adrian Thompson (Sussex) configured one of his experiments directly evolving electronic circuits on a FPGA silicon chip to test the hypothesis that Neutral Networks existed and were used.

Population size 1+1, and only genetic operator (something like|) single mutations.

Search space size 2^{1900}, and an expected number of solutions say roughly 2^{1200}.

Punk Eek



Analysis

Looking at 3 individuals

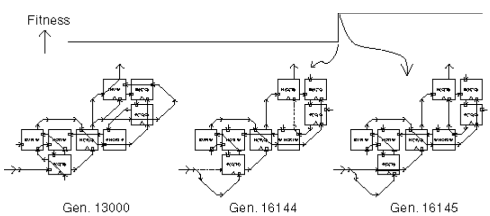
A at beginning of a plateau

B at the end of a plateau

C after the single step up to next plateau

One could check out what happened.

History of a Walk



Was the Walk Necessary ?

Yes.

On checking, none of the possible mutations from A led to higher fitness.

The B->C mutation, if applied to A, actually decreased its fitness.

So the Neutral Network was essential, and was indeed used.

Non Symbolic AI - Lecture 14

EASy

Final Lecture.
Exam/coursework advice + more on Neutral Networks + robot videos

What should you write in the coursework writeup?

EASy

Firstly, your program code should be clear and commented – so the writeup should not just be a further description of the code

Secondly, even if your code does not work as you had hoped, you can still gain marks from a good writeup

Demonstrate your ***understanding!***

EASy

We want to see that you understand the ***basics*** of ANNs, what they can be used for.

The ***basics*** of a learning algorithm such as backprop

The ***basics*** of a GA

The ***basics*** of applying an ANN to a Boolean problem such as this.

Use your own words

EASy

Straight ***repetition*** from the lecture notes or a textbook is a ***bad idea!***

It suggests that you have memorised the words ***without*** understanding them.

Your ***own*** way of describing things will be much more impressive!

Did you think beyond the problem?

EASy

Was the problem a sensible one?

Was backprop a sensible learning algorithm?

Was a GA a sensible idea?

Would you expect all problems to be like this one, or different?

Did you think beyond your results?

EASy

Are you happy with your results? Why?

Were you lucky, or unlucky?

If you did it again once, or many times, would you expect the same answers? Why/why not?

The more intelligent comments that you can make, the better.

Don't leave it to the last minute!

EASy

You should aim to basically finish – and ***print*** out your submission – at least 1, pref 2 days before the deadline.

For some amazing reason, computers and printers break down just before a deadline – it is ***your responsibility*** to anticipate this!

Then with good luck, you may think of some improvements in the last 2 days for a better version – but you are not ***relying*** on good luck.

Exam technique -- Revision

EASy

The Exam is due on [announce date] June.

Pointers to previous exam pages via the course web page.

90 minutes, choose Two out of Three questions.

Third optional question is an essay question, with several possible titles given.

Advice

EASy

Taken from:-

www.informatics.sussex.ac.uk/doc/unseen_exams.php

Revise Non-Sym AI alongside the other courses

A > B > C > NSAI > A > B > C > NSAI > A > B > ….

Revision is done by ***Writing!***

EASy

Take your lecture notes, or textbook or any written material you want to revise – and ***write notes*** in your own words to summarise what is said.

Input Information usually doesn't stick in your brain unless you ***output*** something from it – and writing notes is the best way.

Summarise in notes

EASy

First time round, just summarise a paper or a lecture into just a page or two.

Then start again: see if you can summarise your summary into less than a page.

Then start again: see if you can summarise your new summary into a small postcard – in all this, the ***summarising*** makes sense of it, the ***writing*** makes it stick!

Practice an Exam

EASy

Take an old NSAI exam – lock yourself in a room for 90 minutes – and see how well you do.

Read the questions.

Manage your time.

Don't spend so much time answering your 'favourite question' that you can write only scrappy notes for the other question you choose.

Answering a question

EASy

Plan each answer. Jot down a skeleton answer-plan, on a page which you will cross out as rough work. Especially important for an Essay-answer.

Answer the question on the exam paper - not the one you were expecting to find on the paper.

Check how many points are allocated for each part of a multi-part question – and allocate your time accordingly

Use any spare time to re-read your answers, and improve them

Before the Exam

EASy

Do your revision in good time – finish a couple of days before the exam

Then come out of your hole, take a break, get some exercise, get a good night's sleep before the exam.

… and ***Don't Worry … !***

Bit more on Neutral Networks

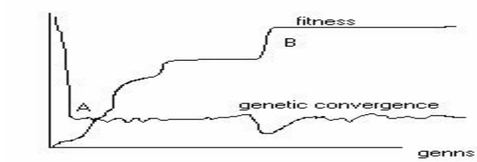
EASy

(Followed by some robot videos)

Basics of Neutral Networks covered in last lecture. Here (at high speed) is a reprise of that, plus a bit more.

Genetic Convergence is not Stasis

EASy



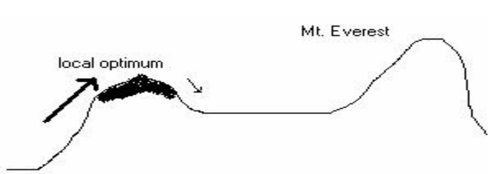
B is the point of convergence (defn 2), often after 'punctuated equilibria'

A is the point of genetic convergence (defn 1), which may well be (surprisingly?) within the first 10 or so generations !

Optimal mutation rates

EASy

Continuing SAGA ideas, in a fitness landscape you can have too little mutation (relative to selection:-

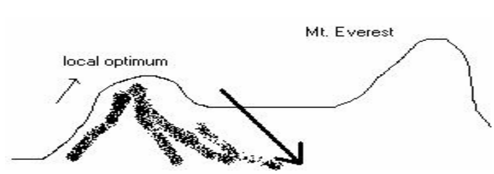


… or too much mutation

EASy

… or you can have too much mutation …

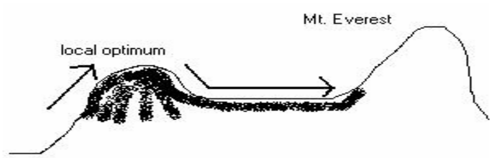
(arrow up the hill is selection, arrow down the hill is mutation)



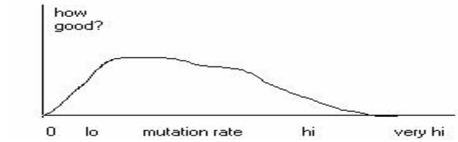
… or just about the right amount …

... or you can gave round about the right amount, to avoid losing height (fitness) gained, but promoting search along **ridges** -- which may lead to higher ground :-

Balance between **exploration** and **exploitation**



Why should there be an optimal mut-rate?



Mutation rate too low, in limit zero, would mean no further change, evolution ceases -> no good

Mutation rate too high, eg every bit flipped at random, implies random search -> no good.

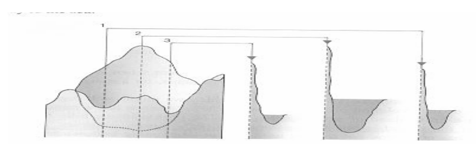
High dimensional landscapes

We can visualise ridges in the 3-D landscapes (Himalayas, South Downs) that the metaphor of fitness landscapes draws upon.

But in 100-D or 1000-D landscapes things can be very significantly different.

In particular you can have ridges in all sorts of directions.

Ridges in high-dimensional landscapes

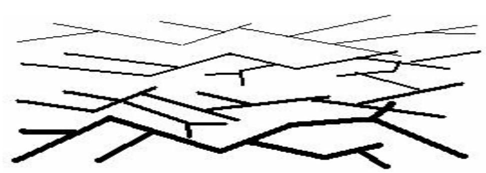


Going from 2-D to 3-D allows extra opportunities for "bypasses around a valley without dropping height"

Going up to 100-D or 1000-D potentially allows many many more such opportunities -- hyper-dimensional bypasses.

(pic borrowed from *Steps Towards Life*, Manfred Eigen Oxford Univ Press 1992)

The New Picture



IF there is lots of neutrality of the right kind, then there are lots of *Neutral Networks*, connected pathways of neutral mutations running through the landscape at one level --

Right kind of redundancy

- Multi-storey building example – some corridors lead to staircases
- When this is so, 'going along the flat' buys you something
- Of course, if corridors lead to flat plains without a staircase in sight, you are wasting your time!

The First claim for Neutral Networks

EASy

(1) **The Formal claim**
It can be demonstrated indisputably that **IF** a fitness landscape has lots of neutrality of a certain kind, giving rise to Neutral Networks with the property of constant innovation

THEN the dynamics of evolution will be transformed (as compared to landscapes without neutrality) and in particular populations will not get stuck on local optima.

The above would be merely a mathematical curiosity unless you can also accept:-

The Second claim for Neutral Networks

EASy

(2) **The Empirical claim**

Many difficult real design problems
(..the more difficult the better...)
in eg evolutionary robotics, evolvable hardware, drug design
--- have fitness landscapes that naturally (ie without any special effort) fit the bill for (1) above.

I make claim (2), but admit it is as yet a dodgy claim!

Recently some supporting evidence.

Recent Research on Neutral Networks

EASy

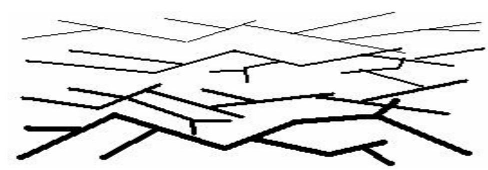
One of the first demonstrations of the formal claim was in an EASy MSc dissertation by Lionel Barnett 1997.

See full dissertation, and shorter version for Alife98 conference, on his web pages
http://www.cogs.susx.ac.uk/users/lionelb/

and Neutral Network bibliography via
http://www.cogs.susx.ac.uk/lab/adapt/nn.html

The New Picture

EASy



IF there is lots of neutrality of the right kind, then there are lots of *Neutral Networks*, connected pathways of neutral mutations running through the landscape at one level --

… percolation …

EASy

-- and *lots and lots* of these NNs, at different levels, **percolating** through the whole of genotype space, passing close to each other in many places.

Without such neutrality, if you are stuck at a local optimum (ie no nbrs higher) then there are only N nbrs to look at *BUT WHEN you have lots of neutrality*, then without losing fitness you can move along a NN, with nearly N new nbrs at every step -- 'constant innovation'.

Basically, you never get stuck !

Right kind of redundancy

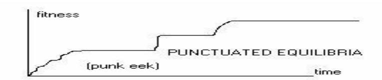
EASy

- Multi-storey building example – some corridors lead to staircases
- When this is so, ‘going along the flat’ buys you something
- Of course, if corridors lead to flat plains without a staircase in sight, you are wasting your time!

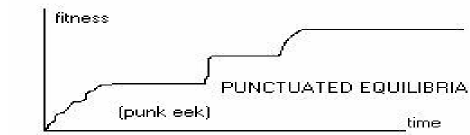
What happens?

Roughly speaking, in such a landscape the population will quickly 'climb onto' a ridge slightly higher than average, then move around neutrally 'looking for a higher nbr to jump to'.

You might have to wait a while (even a long while...) but you will not get stuck for ever. When eventually one of the popn finds a higher NN, the popn as a whole 'hops up‘ and carries on searching as before



Punk Eek



...and significantly, in many real GA problems this is just the sort of pattern that you see.

The horizontal bits are not (as many thought) just standing still waiting for luck --- rather 'running along NNs waiting for luck'

Ruggedness versus Neutrality

Lionel Barnett's NKp landscape gives an abstract framework in which one can tune independently:
K for ruggedness and p for degree of Neutrality.

There are various standard measures for ruggedness e.g. *autocorrelation* -- roughly, a measure of how closely related in height are points 1 apart, 2 apart, ...10 apart...

Amazingly, for fixed N and K, when you tune parameter p all the way from zero neutrality up to maximum neutrality the autocorrelation remains (virtually) unchanged.

Same ruggedness but different dynamics

Yet as you change the neutrality p, despite having the same ruggedness the *evolutionary dynamics changes completely* -- for zero neutrality the population gets easily stuck on local optima, for high neutrality it does not.

Clearly neutrality makes a *big difference* -- yet this has been completely unknown to the GA community, who have only worried about ruggedness.

Indeed all the typical benchmark problems used to compare different GAs have no neutrality at all.

Net-crawlers

- The EH example was basically a bastardised version of a GA – a **net-crawler** – equivalent to a Steady-state GA with population size 2.
- Lionel Barnett, in his thesis, showed that for a particular class of abstract fitness landscape (epsilon-correlated) that had many NNs, s.t. that the population could jump from one to the next, then **provably** the best search method was a net-crawler with a specific rate of mutation

Net-crawling mutation rate

- Optimal rate is **provably** (under certain assumptions):-
- Mutate exactly n (an integer) bits on genotype
- … where n is chosen so as to make the percentage of neutral mutations as close as possible to 37% (1/e)
- (using plausible assumptions) this can be calculated on the fly, keeping track of how many recent mutations were neutral and adjusting mutation up/down accordingly
- For the EH example, this looks like suggesting 3 mutations !!!

Summary on NNs

Please distinguish between

1. The **FORMAL claim**, proven without doubt: that fitness landscapes full of neutral networks of the right kind completely alter evolutionary dynamics
2. The **EMPIRICAL claim,** that many real-world difficult design problems, with (near-)binary encodings, do in fact have lots of neutral networks of the right kind.

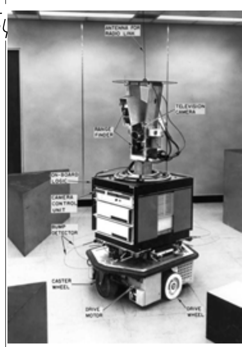
Classical AI

When building robots, the Classical AI approach has the robot as a scientist-spectator, seeking information from outside.

"SMPA" -- so-called by Brooks (1999)

- S sense
- M model
- P plan
- A action

Shakey

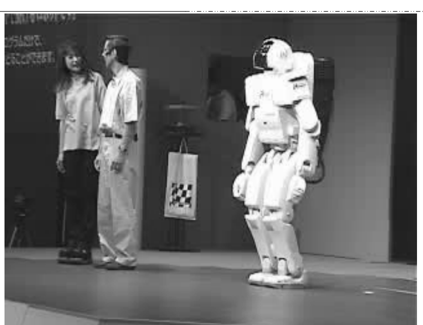


1970-Shakey the robot reasons about its blocks
Built at Stanford Research Institute, Shakey was remote controlled by a large computer. It hosted a clever reasoning program fed very selective spatial data, derived from weak edge-based processing of camera and laser range measurements. On a very good day it could formulate and execute, over a period of hours, plans involving moving from place to place and pushing blocks to achieve a goal.

Courtesy of SRI International.

The Honda Humanoid Robot

Honda



Brooks' alternative

Brooks' alternative is in terms of many individual and largely separate **behaviours** – where any one behaviour is generated by a pathway in the 'brain' or control system all the way from Sensors to Motors.

No Central Model, or Central Planning system.

Sojourner Rover on Mars

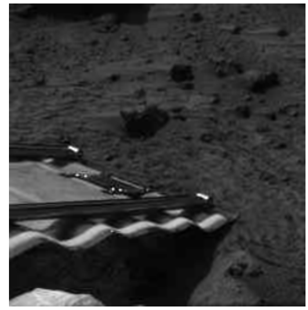
NASA and JPL

July 1997

Based heavily on Brook's ideas

"Fast cheap and out of control!"

Sojourner



Sojourner

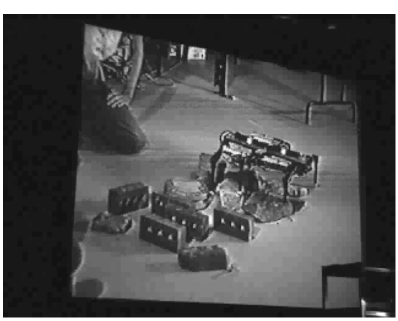
Semi-autonomous

Signals from Earth took around 30 minutes to reach Mars

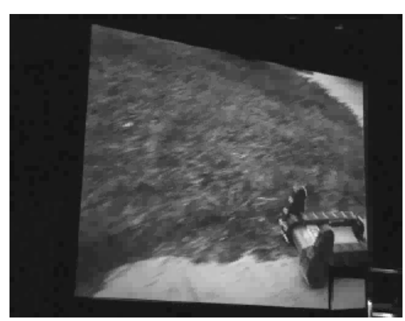
Subsumption summary

- New philosophy of hand design of robot control systems
- Incremental engineering – debug simpler versions first
- Robots must work in **real time** in the **real world**
- Spaghetti-like systems unclear for analysis
- Not clear if behaviours can be re-used
- Scaling – can it go more than 12 behaviours?

Over rough terrain

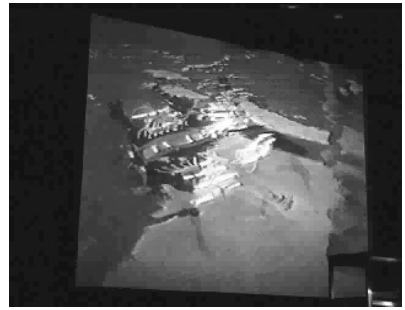


For military or rescue



For military or rescue

Underwater



THE END !

Non-Symbolic AI Guest lecture

EASy

This will be a Guest Lecture by Eric Vaughan on Passive Dynamic Walking, including very current research.