THE UNIVERSITY of York

Evolving an artificial brain to solve multiple problems

Julian F. Miller
Department of Electronic Engineering
University of York
julian.miller@york.ac.uk

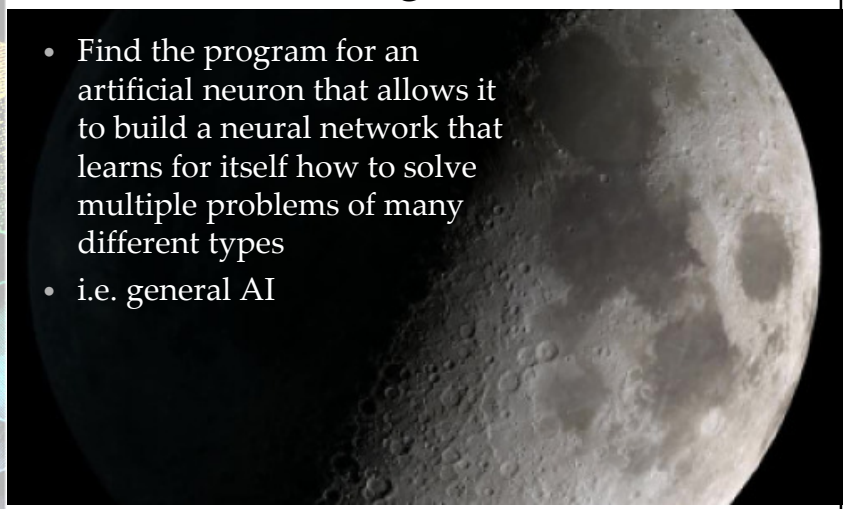
DevoNN

1

THE UNIVERSITY of York

Moonshot challenge

- Find the program for an artificial neuron that allows it to build a neural network that learns for itself how to solve multiple problems of many different types
- i.e. general AI



2

THE UNIVERSITY of York

Overview

- Biological Development is responsible for all learning in brains
  - Brains solve *multiple problems* without interference
  - In brains, neurons move, change, die or replicate, dendrites and axons change, branch, shrink and die
  - Learning is not just synaptic but *topological*
  - Learning happens in the lifetime of the individual
- Almost all ANN models do not have development
  - Aim: Find a set of *weights* for *fixed* connections that give good performance ("synaptic dogma")
  - They suffer from catastrophic interference

3

THE UNIVERSITY of York

The Neuron model: overview 1

- Neuron abstracted as a body (soma) and a number of connections (dendrites)
- The collection of neurons and dendrites is called the ``brain''
- There are two evolved programs: soma and dendrite
  - When executed in some initial neurons, a brain grows
    - The soma program allows neurons to move, die, replicate, or change
    - The dendrite program allows the dendrites to change in length, die, replicate , or change
- All neural variables are real-valued in the interval [-1.0, 1.0]
- The soma and dendrite programs are encoded and evolved using Cartesian GP
  - Nodes in the CGP graph are mathematical functions they operate on and return values between -1 and 1.

4

The Neuron model: overview 2

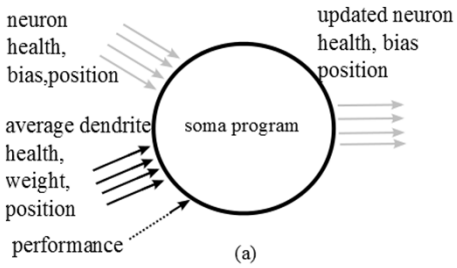
Some neurons provide the *outputs* corresponding to a computational problem

Inputs can be shared or unshared and are assigned fixed positions which are problem dependent

From the "brain" traditional feedforward ANNs are extracted

Soma program

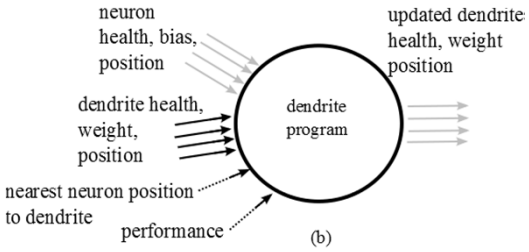
- Neurons have a bias, health and a position



- The program adjusts neuron variables (health, bias and position): if they are positive (negative) they are incremented (decremented) by user-defined increments: δ_{sh}, δ_{sb}, δ_{sp}
- Non-output neurons have performance input equal to zero. Output neurons have performance input equal to `fitness' score on the problem being solved

Dendrite program

- Dendrites have a health, weight and position
- The dendrite program is run in *every* dendrite



- The program adjusts neuron variables (health, bias and position): if they are positive (negative) they are incremented (decremented) by user-defined increments: δ_{dh} , δ_{dw} , δ_{dp}
- Dendrites on non-output neurons receive a performance score of zero. Dendrites on output neurons receive the performance score given to the parent neuron

Birth, death and change

- There are user defined health thresholds for neurons (HN_l , HN_u) and dendrites (HD_l , HD_u) which determine whether a neuron or dendrite dies, changes or replicates
  - $h_i < HN_l$: neuron death $h_i > HN_u$: neuron birth
  - $h_{ij} < HD_l$: dendrite death $h_{ij} > HD_u$: dendrite birth
    - Neuron health decides dendrite birth
- If no birth or death, the neuron or dendrite changes according to the updated values from the neuron program
  - When birth occurs a new neuron is placed at a distance MN_{inc} to the right of the parent neuron.
  - If neuron collisions occur, neurons are moved by MN_{inc} until no more collisions occur
- Neuron and dendrite numbers are bounded between one and the maximum allowed (user defined)

8

```
1: function FITNESS
2:     Initialise brain
3:     Load 'pre' development parameters
4:     for NDS_pre times do
5:         Run soma/dendrite programs to update brain
6:     end for
7:     Load 'while' developmental parameters
8:     Epoch = 0
9:     PreviousFitness = 0
10:    repeat
11:        for NDS_whi times do
12:            Run soma/dendrite programs to update brain
13:        end for
14:        for p = 0 to NumBenchmarkProblems do
15:            Extract ANN
16:            if IncrementalTraining then
17:                N_incr = (epoch + 1) * NTr(p)/N_ep
18:                for N_incr training cases do
19:                    Accumulate fitness
20:                end for
21:                Fitness = Fitness/N_incr
22:            end if
23:            Accumulate Fitness for benchmarks
24:        end for
25:        Fitness = Fitness/NumBenchmarkProblems
26:        if EpochTraining then
27:            if Fitness < PreviousFitness then
28:                Break
29:            else
30:                PreviousFitness = Fitness
31:            end if
32:        end if
33:        Increment Epoch
34:    until N_ep epochs
35:    Return Fitness
36: end function
```

← Development without learning

← Development in learning loop
May have fitness input

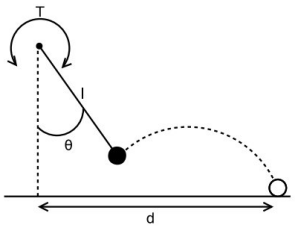
← Increase training set

← If epoch training look for
higher fitness at each epoch

Task problems

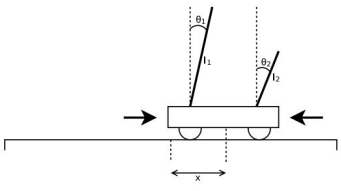
Ball throwing

- Inputs: Arm angle, angular velocity (2)
- Outputs: Torque, Ball Release (2)
- Fitness: distance thrown (max 10.202)
- Upto 10,000 simulation steps



Double-pole balancing

- Inputs: Arm angle, angular velocity of pole, position and velocity of cart (6)
- Outputs: Force applied
- Fitness: how long balanced
- Upto 100,000 simulation steps



Results 1

One problem

Statistic	Ball throw	Double pole
Mean	0.8515	0.8455
Median	0.9827	1
Maximum	0.9948	1
Minimum	0.5394	0.00278
No. solved	14	12

Two problems

Statistic	Experiment A NN position used. Incr. training Performance input No epoch training	Experiment B NN position unused. Incr. training Performance input No epoch training
Mean	**0.4072**	0.3357
Median	**0.2741**	0.2735
Maximum	**0.7728**	0.7372
Minimum	0.2701	0.2702
Max Prob 1	0.5458	**0.6699**
Max Prob 2	**1.0 (4)**	0.9347 (0)

- Nearest neighbour (NN) position useful to dendrite
- Also found that incremental learning highly beneficial
- Solving ball-throw and double-pole simultaneously, is much harder than solving either
  - Suggests interference is occuring…

Results 2: Shared v. unshared inputs (not in paper)

Statistic	Experiment A Shared inputs NN position used. Incr. training Performance input No epoch training	Experiment E Unshared inputs NN position used. Incr. training Performance input No epoch training
Mean	0.4072	**0.4149**
Median	0.2741	**0.4845**
Maximum	**0.7728**	0.7614
Minimum	0.2701	**0.2705**
Max Prob 1	0.5458	**0.9980 (11)**
Max Prob 2	**1.0 (4)**	0.5372

- Using unshared inputs appear to give better results and is conceptually more acceptable

THE UNIVERSITY of York

Results 3: Solving both classification and reinforcement learning problems (ball throwing) (not in paper)

Statistic	Cancer classification training	Cancer classification test	Ball-throwing
Mean	0.9410	0.8300	0.7891
Median	0.9471	0.9224	0.9378
Maximum	0.9600	0.9598	0.9926
Minimum	0.8886	0.4655	0.5310
Num solved	-	-	11

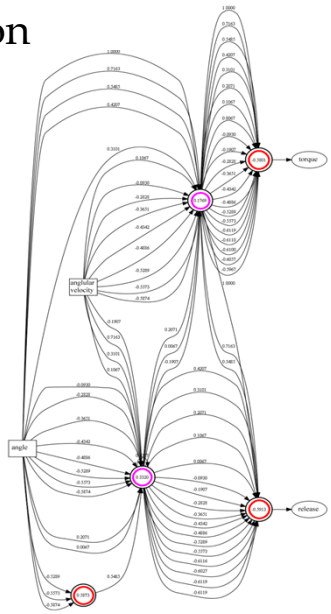
- It is relatively easy to evolve the neural programs to produce a brain that can solve both classification and a reinforcement learning problem

13

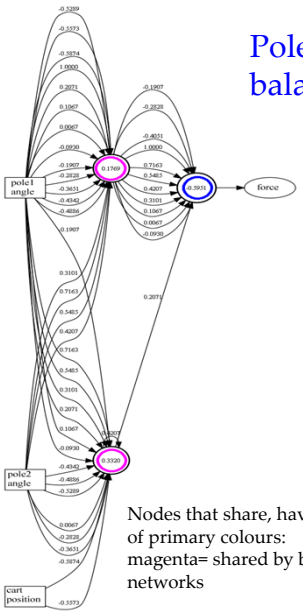
THE UNIVERSITY of York

A perfect solution

Ball throwing



Pole balancing



Nodes that share, have mixtures of primary colours: magenta= shared by both networks

14

THE UNIVERSITY of York

Future work

- Incremental problem solving
  - Doesn't seem to help. Why?
- Increase the number of problems
  - What are suitable problems?
- Favour best learners rather than superb individuals?
- Decrease interference
  - Maybe output neurons should have their own parameters distinct from non-output neurons?
- What are appropriate inputs and outputs for a developmental ANN?
  - We need something universal that applies to all problems
- Eventually introduce spiking neurons

15

THE UNIVERSITY of York

Conclusions

- Evolving developmental programs using GP to build ANNs is exciting but *very* challenging
  - Very few researchers are working on this…
    - Many are falling into the HyperNEAT attractor
- Many decisions have to be made about small details
- Biological neurons are extremely complex, artificial devANNs are so different it is hard to know what aspects of biological neurons are essential/useful in devANNs
- Please consider joining this moonshot challenge. We need you!

16