NEAT

01710501068

Jadavpur University

February, 2021

Neuro-Evolution

Neuro-Evolution is the practice of generating Artificial Neural Networks using evolutionary algorithms. This might involve tuning only the weights (parameters) of a network with the topology decided beforehand. Or both the topology and the connection weights might be evolved using a genetic algorithm. This can be used to solve control tasks by using a measure of the network's performance for the task as fitness. So it presents an alternative to statistical methods which attempt to estimate particular actions in particular states of the world (reinforcement learning).

Two main types of neuro-evolution

- conventional fixed-topology (only evolve weights)
- TWEANN (topology and weight evolving artificial neural network)

- **Moriarty and Miikkulainen, 1996** showed that neuro-evolution was faster than reinforcement learning in specific tasks like **single pole balancing** and **robotic arm movement**.
- **Chen et al., 1993** has shown modifying the network structure can be effective as part of supervised training.
- **Gruau et al., 1996** claimed that time is wasted on deciding the no. of hidden nodes by trial and error, so evolving topology would be more efficient. He showed it on a pole balancing task.
- **Gomez and Miikkulainen, 1999** showed that structure was not important for that task. It used a method called Enforced Subpopulations and solved the problem 5 times faster.
- **Cybenko, 1989** has shown a fully connected network can approximate any continuous function.

because NE searches for a behavior instead of a value function, it is effective in problems with continuous and high-dimensional state spaces. but can evolving structure provide any advantage ?

Neuro Evolution of Augmenting Topologies (NEAT) is a method that wants to minimize the no. of dimensions of the search space over connection weights by evolving the structure of the neural network along with the weights.
They want to start with a minimal network and only keep stuctures that are useful.

Encoding

how would the neural network be representated as a chormosome ?

earlier attempts

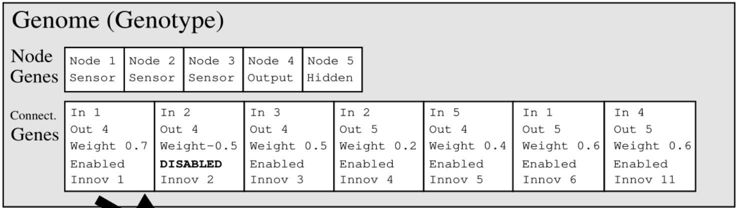
- binary encoding: the connection matrix of the network is representated using a bit string. [sGA]
  the no. of bits is required is the square of the no. of nodes so the size of chromosome increases quickly.
- graph encoding: a grid representing the adjacency matrix followed by a genome of node definitions specifying incoming and outgoing connections. [PDGP]

if the no. of nodes is fixed, the 2D represntation would allow easy **subgraph swapping**. swap all the connections for a particular node by swapping the row between the matrices.

Encoding

NEAT

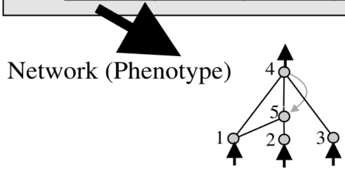
- node genes: a list of input, output and hidden nodes that can be connected.
- connection genes: specifies the in-node, out-node, an innovation no. and an enable bit.



Genome (Genotype)

Node Genes	Node 1 Sensor	Node 2 Sensor	Node 3 Sensor	Node 4 Output	Node 5 Hidden

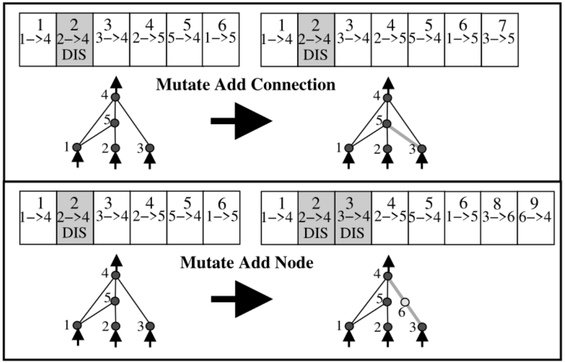
Connect. Genes	In 1	In 2	In 3	In 2	In 5	In 1	In 4
	Out 4	Out 4	Out 4	Out 5	Out 4	Out 5	Out 5
	Weight 0.7	Weight-0.5	Weight 0.5	Weight 0.2	Weight 0.4	Weight 0.6	Weight 0.6
	Enabled	**DISABLED**	Enabled	Enabled	Enabled	Enabled	Enabled
	Innov 1	Innov 2	Innov 3	Innov 4	Innov 5	Innov 6	Innov 11



Mutation

Mutation can change both connection weights and network structures.

- connection weights: each one is changed or kept, based on a probability check.
- network structure
  - **add connection:** a single connection gene with random weight is added connecting 2 previously unconnected nodes.
  - **add node:** a connection is split. first a new node gene is added. then an old connection is disabled, and 2 new connections are added in it's place going through the new node. the connection leading into the new node receives weight=1 and the connection leading out receives the weight of the old(disabled) connection.



1	2	3	4	5	6
1–>4	2–>4	3–>4	2–>5	5–>4	1–>5
	DIS				

1	2	3	4	5	6	7
1–>4	2–>4	3–>4	2–>5	5–>4	1–>5	3–>5
	DIS					

1	2	3	4	5	6
1–>4	2–>4	3–>4	2–>5	5–>4	1–>5
	DIS				

1	2	3	4	5	6	8	9
1–>4	2–>4	3–>4	2–>5	5–>4	1–>5	3–>6	6–>4
	DIS	DIS					

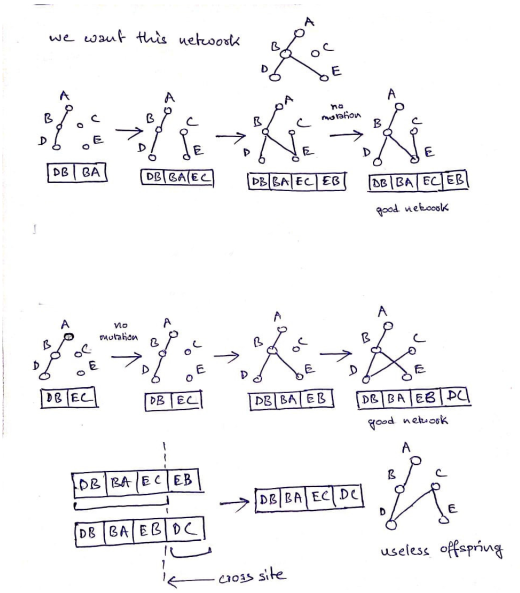
Competing Conventions

Competing conventions means having more than one way to express a solution to a weight optimization problem with a neural network. When genomes representing the same solution do not have the same encoding, crossover is likely to produce damaged offspring.
The same problem is present in nature as well. During reproduction if any gene can insert itself in any position on the genome without any indication of which gene is which, life probably would not have succeeded. In E.coli a protein lines up the homologous genes between 2 genomes before cross-over occurs. So that the right genes can be replaced with the right genes.

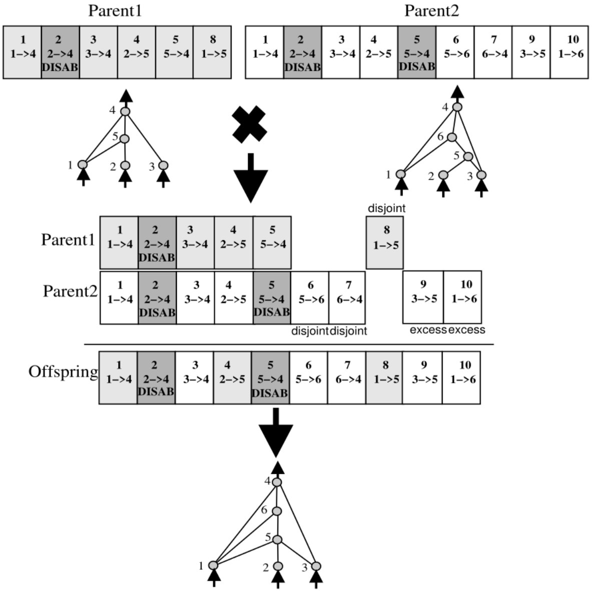
Historical Origin

NEAT tracks the historical origin of each gene in order to line up the genes representing the same structure during cross-over. Two genes with the same historical origin must represent the same structure (with possibly different weights). Whenever a new gene is created from mutation, an innovation number is incremented and assigned to the gene. The innovation numbes represent a chornology in which the genes were added.
When the same structural innvoation occurs through 2 different mutations in the same generation they are given the same innovation number. To prevent expolsion of innvoation numbers.

Crossover Problem



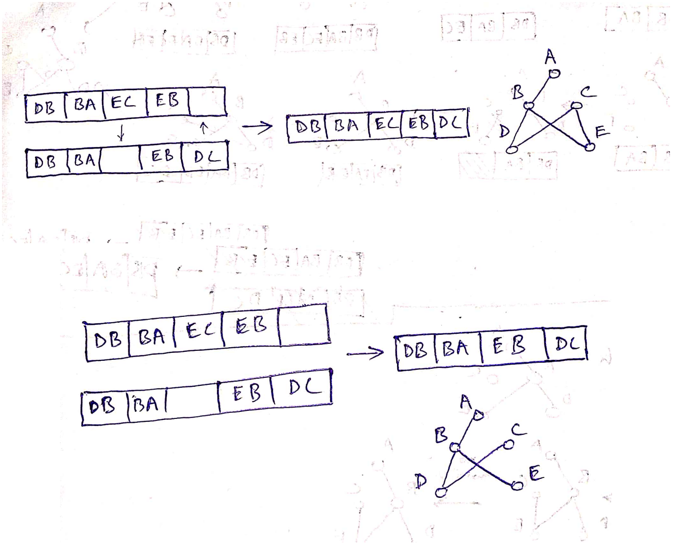
Crossover


Parent1
1 1->4
2 2->4 DISAB
3 3->4
4 2->5
5 5->4
8 1->5
Parent2
1 1->4
2 2->4 DISAB
3 3->4
4 2->5
5 5->4 DISAB
6 5->6
7 6->4
9 3->5
10 1->6
disjoint
disjoint
disjoint
excess
excess
Offspring
1 1->4
2 2->4 DISAB
3 3->4
4 2->5
5 5->4 DISAB
6 5->6
7 6->4
8 1->5
9 3->5
10 1->6
1
2
3
4
5
6

Crossover

When crossing over the genes with same innovation numbers are lined up. Genes that do not line up are called disjoint or excess based on whether the occur within or outside the range of the other parent's innovation numbers. They represent structure that is not present in the other genome.

Crossover



DB
BA
EC
EB
DB
BA
EB
DC
DB
BA
EC
EB
DC
A
B
C
D
E
DB
BA
EC
EB
DB
BA
EB
DC
DB
BA
EB
DC
A
B
C
D
E

Mutation Problem

When a new node or connection is added the fitness of the network would often be reduced. Because it is unlikely that a newly added connection with a random weight would express a useful function. Due to the loss of fitness the innovation is unlikely to survive long enough for the weights optimized. So it is necessary to have some mechanism to protect it.

In nature different structures tend to be in different species that compete in different niches. So innovations are protected within a niche. Similarly networks with innovations can be isolated into their own species.

To do this we would need some measure of compatibility between networks and a threshold to determine which networks should belong to the same species.

Speciation

The no. of disjoint and excess genes between a pair of genomes is a natural measure of their compatibility distance.
So the distance is calculated as a linear combination of the no. of excess genes E, disjoint genes D, and the average weight differnce of matching genes. N = no. of genes in the larger genome.

$$\delta = \frac{c_1 E}{N} + \frac{c_2 D}{N} + c_3 \cdot \overline{W}.$$

Speciation

- Each existing species would be represented by a random genome inside the species from the previous generation.
- A genome g in the current generation is placed in the first species with which it is compatible.
- they are using explicity fitness sharing. meaning the fitness of an individual is scaled down by the no. of individuals in the same species.

$$f_i' = \frac{f_i}{\sum_{j=1}^{n} \mathrm{sh}(\delta(i,j))}.$$

Survival

- each species is given a quota of no. of offspring in the population based on the average fitness of the species.
- no. of offspring = (average fitness of a species)*(total population size)/(sum of average fitnesses of all species)
- the lowest performing members of each species are eliminated and the offspring of the remaining the members are used to replace the population. The no. offspring would be divided among the remaining members of each species.
- the champion of a species with more than 5 networks was carried over unchanged.
- when the fitness did not increase for more than 20 generations only the top 2 species were allowed to reproduce.

Initial Population

- other neuro-evolution methods typically start with an initial population of random topologies in order to introduce diversity from the start.
- NEAT focuses the search towards lower dimensional spaces by reducing the network structure. It starts with a uniform population of networks with 0 hidden nodes. New structure is added incrementally through mutations and only those structures survive that are found to be useful from fitness evaluations.

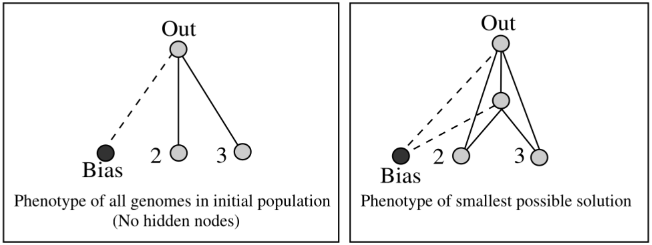
Parameters

- population size 150
- c1 = 1.0, c2 = 1.0
- c3 = 0.4 to allow for finer distinction based on weight differences
- 80% chance for weights to be mutated
  - 90% chance to be uniformly perturbed, 10% chance to be assigned new random value.
- 75% for an inherited gene to be disabled if it was disabled in either parent
- 25% of offspring resulted from mutation without crossover
- inter-species mating rate 0.001
- probability of adding a new node was 0.03
- probability of adding a new link was 0.05 [7.5]
- $\varphi(x) = \frac{1}{1+e^{-4.9x}}$

XOR verification

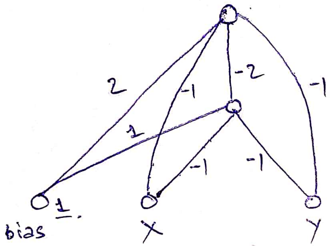
XOR is not linearly separable so a neural network requires hidden nodes to calculate it. It can be used to test NEAT's ability to evolve structure.

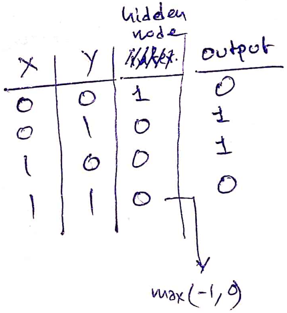
- the input layer consisted of 2 input variables and 1 bias (always 1). there was no hidden node in the initial network and 1 output node.
- there were 3 connection genes in each member of the inital population. each connecting a input/bias node to the output. each with a random weight.
- fitness was calculated as
  4 - (sum of distance from ans for all 4 input combinations)
  so that higher fitness would mean better network.


Out
Bias
2
3
Phenotype of all genomes in initial population
(No hidden nodes)
Out
Bias
2
3
Phenotype of smallest possible solution

(a) (b)

XOR verification



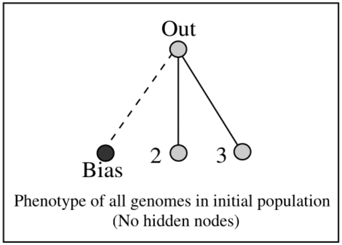


x	y	hidden node ~~[illegible]~~	output
0	0	1	0
0	1	0	1
1	0	0	1
1	1	0	0

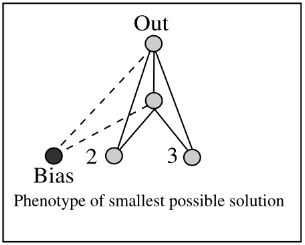
max(-1, 0)

XOR verification

- 100 experimental runs were performed
- NEAT finds a structure for XOR in an average of 32 generations.
- On average the solution network had 2.35 hidden nodes and 7.48 non-disabled connection genes.
- The optimal solution (with a single hidden node) was found in 22 of the 100 runs.



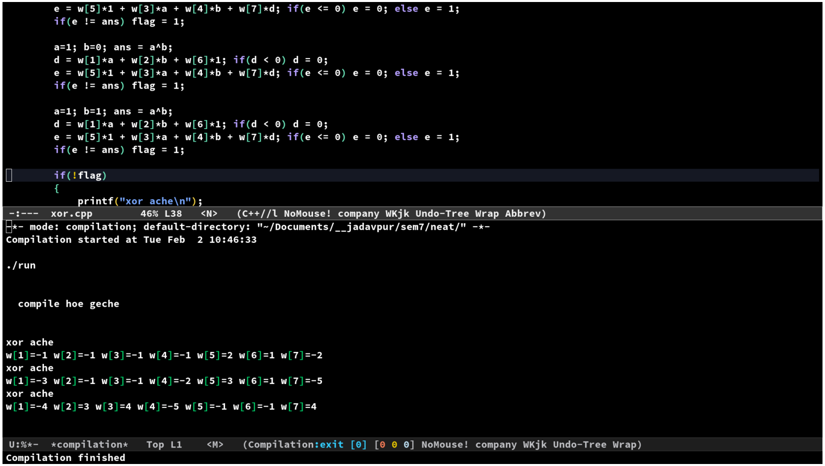
(a)



(b)

XOR verification

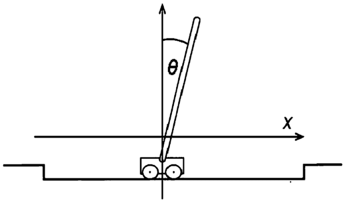
rand()%10 - 5



```
        e = w[5]*1 + w[3]*a + w[4]*b + w[7]*d; if(e <= 0) e = 0; else e = 1;
        if(e != ans) flag = 1;

        a=1; b=0; ans = a^b;
        d = w[1]*a + w[2]*b + w[6]*1; if(d < 0) d = 0;
        e = w[5]*1 + w[3]*a + w[4]*b + w[7]*d; if(e <= 0) e = 0; else e = 1;
        if(e != ans) flag = 1;

        a=1; b=1; ans = a^b;
        d = w[1]*a + w[2]*b + w[6]*1; if(d < 0) d = 0;
        e = w[5]*1 + w[3]*a + w[4]*b + w[7]*d; if(e <= 0) e = 0; else e = 1;
        if(e != ans) flag = 1;

        if(!flag)
        {
            printf("xor ache\n");
-:---  xor.cpp        46% L38    <N>    (C++//l NoMouse! company WKjk Undo-Tree Wrap Abbrev)
-*- mode: compilation; default-directory: "~/Documents/__jadavpur/sem7/neat/" -*-
Compilation started at Tue Feb  2 10:46:33

./run


  compile hoe geche


xor ache
w[1]=-1 w[2]=-1 w[3]=-1 w[4]=-1 w[5]=2 w[6]=1 w[7]=-2
xor ache
w[1]=-3 w[2]=-1 w[3]=-1 w[4]=-2 w[5]=3 w[6]=1 w[7]=-5
xor ache
w[1]=-4 w[2]=3 w[3]=4 w[4]=-5 w[5]=-1 w[6]=-1 w[7]=4

U:%*-  *compilation*    Top L1     <M>    (Compilation:exit [0] [0 0 0] NoMouse! company WKjk Undo-Tree Wrap)
Compilation finished
```

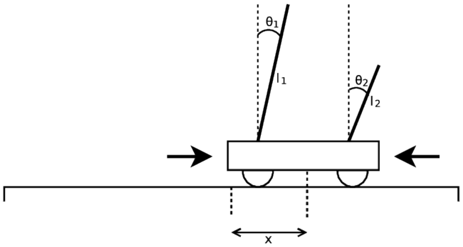
single pole balancing

- a pole is attached to a cart by a hinge and the neural network must apply force to the cart to keep the pole balanced for as long as possible without going beyond the boundaries of the track.
- the position of the cart, position(angle) of the pole, velocity of cart, (angular)velocity of pole are input at each state.
- this version has become easy for modern methods



double pole balancing

- 2 poles are attached to a cart by a hinge and the neural network must apply force to the cart to keep the poles balanced for as long as possible without going beyond the boundaries of the track.
- the position of the cart, position(angle) of the poles, velocity of cart, (angular)velocities of pole are input at each state.



double pole balancing

- All state variables were scaled to [-1.0, 1.0] before being fed to the network
- Networks output a force every 0.02 seconds between [-10, 10]N
- The poles were 0.1m and 1.0m long
- The initial position of the long pole was 1 degree and the short pole was upright
- the track was 4.8 meters long
- 2 versions
    - with velocities
    - without velocities

double pole balancing with velocity

- pole is considered balanced between -36 and 36 degrees.
- fitness was measured as no. of time-steps that both the poles remained balanced.
- NEAT used less hidden nodes on average than fixed-topology methods.

double pole balancing without velocity

- velocities would not be provided as part of input
- non-markovian task
- an internal state has to be kept to estimate velocity information
- pole is considered balanced between -36 and 36 degrees
- special fitness function was created to penalize oscilliations

$$f_1 = t/1000,$$

$$f_2 = \begin{cases} 0 & \text{if } t < 100, \\ \frac{0.75}{\sum_{i=t-100}^{t}(|x^i|+|\dot{x}^i|+|\theta_1^i|+|\dot{\theta}_1^i\ |)} & \text{otherwise.} \end{cases}$$

- t $=$ no. of time-steps both the poles remained balanced during 1000 total time-steps
- Fitness $=$ 0.1 f1 $+$ 0.9 f2

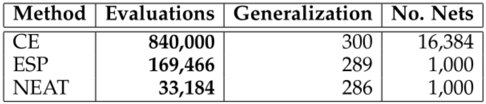
double pole balancing without velocity

Under Gruau et al's criteria for a solution, the champion of each generation is tested on generalization to make sure it is robust. This test takes a lot more time than the fitness test, which is why it is applied only to the champion. In addition to balancing both poles for 100,000 time steps, the winning controller must balance both poles from 625 different initial states, each for 1,000 times steps. The number of successes is called the generalization performance of the solution. In order to count as a solution, a network needs to generalize to at least 200 of the 625 initial states.

initial states are created by giving each of the input variables (cart position, cart velocity, long pole angle, long pole angular velocity) 0.05, 0.25, 0.5, 0.75, 0.95 scaled to the range of the input variable.

double pole balancing without velocity

- NEAT takes 25 times fewer evaluations than Gruau's original benchmark, showing that the way in which structure is evolved has significant impact on performance
- NEAT is also 5 times faster than ESP, showing that structure evolution can indeed perform better than evolution of fixed topologies



Method	Evaluations	Generalization	No. Nets
CE	**840,000**	300	16,384
ESP	**169,466**	289	1,000
NEAT	**33,184**	286	1,000

double pole balancing without velocity

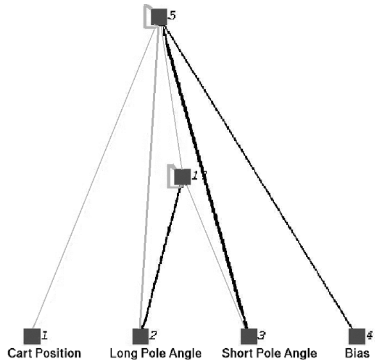
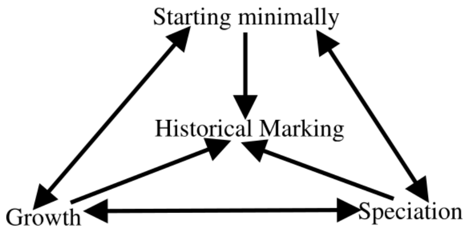


Figure 8: A NEAT solution to the DPNV problem. This clever solution works by taking the derivative of the difference in pole angles. Using the recurrent connection to itself, the single hidden node determines whether the poles are falling away or towards each other. This solution allows controlling the system without computing the velocities of each pole separately. Without evolving structure, it would be difficult to discover such subtle and compact solutions. Starting minimally makes discovering such compact solutions more likely.

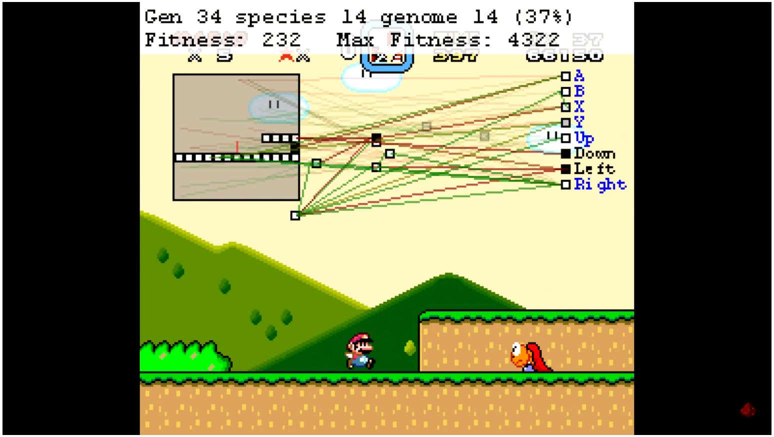
ablation

Method	**Evaluations**	**Failure Rate**
No-Growth NEAT (Fixed-Topologies)	**30,239**	80%
Nonspeciated NEAT	**25,600**	25%
Initial Random NEAT	**23,033**	5%
Nonmating NEAT	**5,557**	0
Full NEAT	**3,600**	0



MarI/O

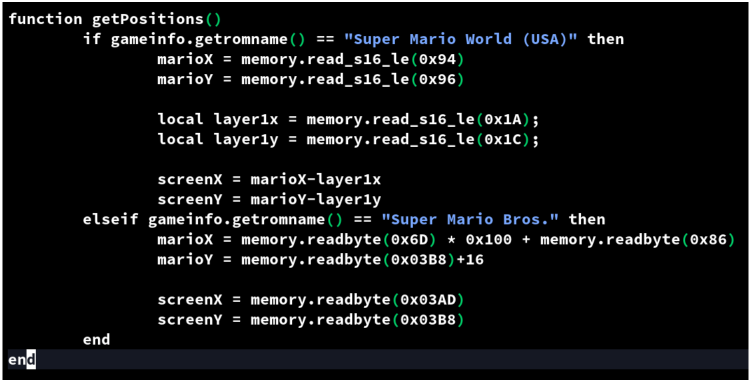
- NEAT was used to succesfully train a neural network to play a level of super mario world.
- progress through the level was used as fitness value.


Gen 34 species 14 genome 14 (37%)
Fitness: 232 Max Fitness: 4322

MarI/O

fitness

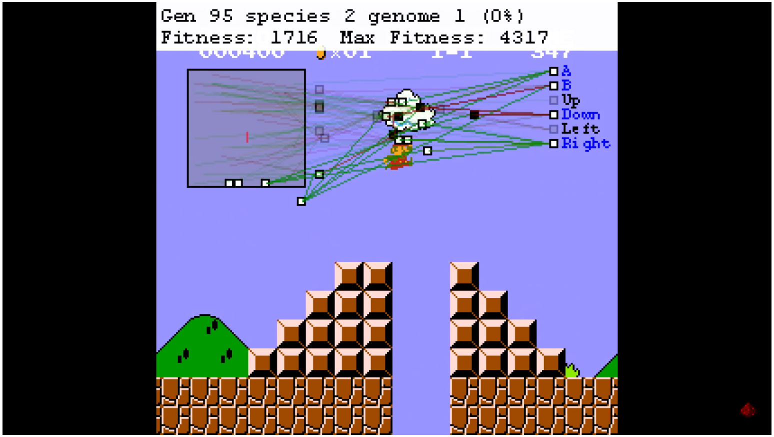
the x co-ordinate of the player was read directly from the emulated memory. the exact bytes being used to store mario's co-ordinates had to be identified.



```
function getPositions()
        if gameinfo.getromname() == "Super Mario World (USA)" then
                marioX = memory.read_s16_le(0x94)
                marioY = memory.read_s16_le(0x96)

                local layer1x = memory.read_s16_le(0x1A);
                local layer1y = memory.read_s16_le(0x1C);

                screenX = marioX-layer1x
                screenY = marioY-layer1y
        elseif gameinfo.getromname() == "Super Mario Bros." then
                marioX = memory.readbyte(0x6D) * 0x100 + memory.readbyte(0x86)
                marioY = memory.readbyte(0x03B8)+16

                screenX = memory.readbyte(0x03AD)
                screenY = memory.readbyte(0x03B8)
        end
end
```

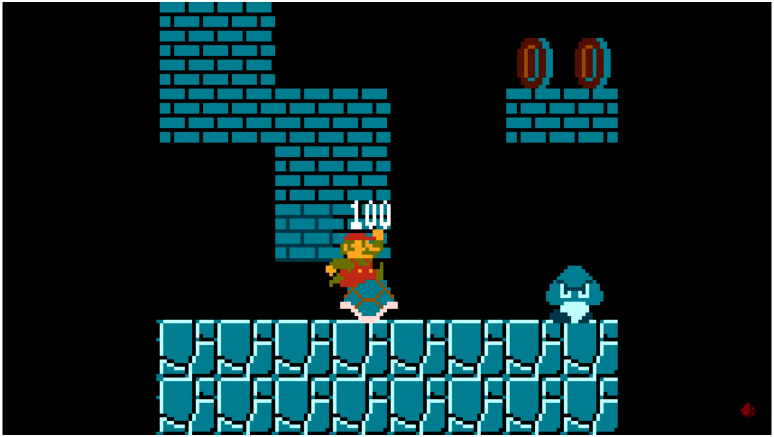
MarI/O

- the network was able to beat the first level of super mario brothers


Gen 95 species 2 genome 1 (0%)
Fitness: 1716 Max Fitness: 4317
A
B
Up
Down
Left
Right

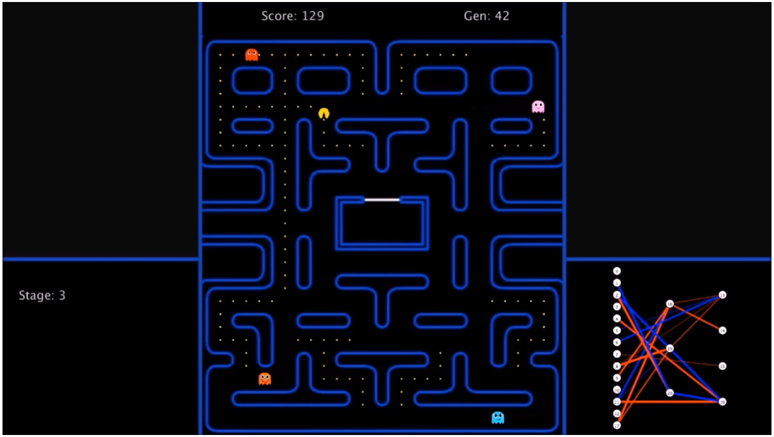
MarI/O

- in the 2nd level it knew a single frame jump was possible



pacman

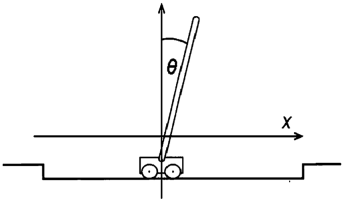
- NEAT was also used to train an AI to finish pacman (eat all the dots)



variations

- rtNEAT
- HyperNEAT
- cgNEAT
- odNEAT

- neuroevolution
- history
- encoding
- mutation
- competing conventions
- cross-over
- speciation
- survival
- initial population
- xor verification
- single pole balancing
- double pole balancing
- double pole balancing without velocity
- mario
- pacman
- comparison



reference

- Kenneth O. Stanley, Risto Miikulainen Evolving Neural Networks through Augmenting Topologies, Evolutionary computation, 2002
- MarI/O