Evolving Neural Networks to Play Checkers

- Can we build intelligent systems to **learn** to play checkers ?
- No expert knowledge provided to the learning system.
- Programs simply have to play against themselves, and figure out how to play.
- Evolutionary computation feasible approach ?

Game playing

- Board representation
- Move search: minimax algorithm
- Traditional game playing programs: board evaluation functions are extensively knowledge based
  1. weighted feature function
  2. opening games
  3. end games table look-up

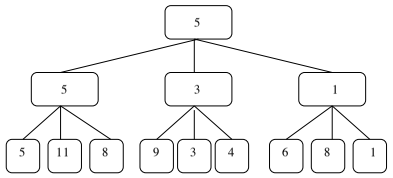
$\Rightarrow$ they do not learn by themselves !

Board representation

- Output $\in [-1 \ldots +1]$
  -1: loss positions
  $+1$: win positions
  $\rightarrow$ closer to $+1 \Rightarrow$ better evaluations
- Input: vector of 32 possible positions, 5 possible values
  1. - K : king opponent
  2. - 1 : checker opponent
  3. 0 : empty
  4. + 1 : checker self
  5. + K : king self

  K $\in [1 \ldots 3]$: exact value evolved

Move decision: mini-max search



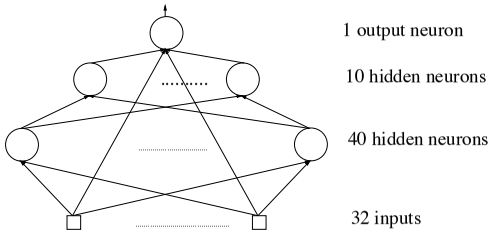
Self move: maximize

Opponent move: minimize

Game lookahead

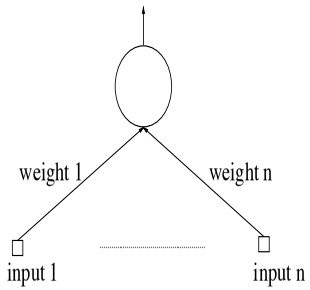
- The further we can lookahead the better.
- Computational restrictions: number of possible board positions grows very fast with increasing number of lookahead levels.
- Deep Blue when defeating chess champion Garry Kasparov made 200 million chess board evaluations per second !
- Here only lookahead search of 2 moves each side when evolving.
- When testing against players on Internet: lookahead search of 3 moves each side.

Board Evaluation



- Evaluation function represented by an artificial neural network
- Use evolutionary algorithm to evolve suitable neural network

Artificial neuron

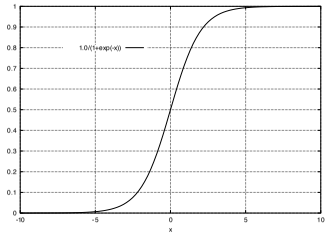


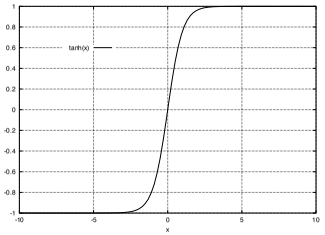
$$x_i, w_i, b, y \in \mathbb{R} : y = tanh(\sum_{i=1}^{n} w_i x_i + b)$$

Sigmoid activation functions

$$\sigma(X) = \frac{1}{1 + exp(-X)} \qquad \sigma(X) = tanh(X)$$

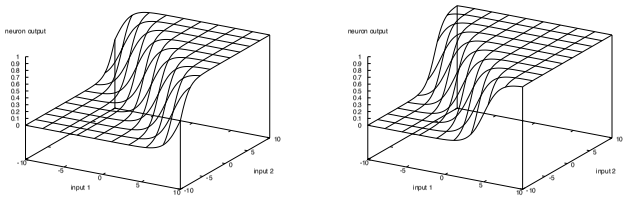
$$0 < \sigma(X) < 1 \qquad -1 < \sigma(X) < 1$$





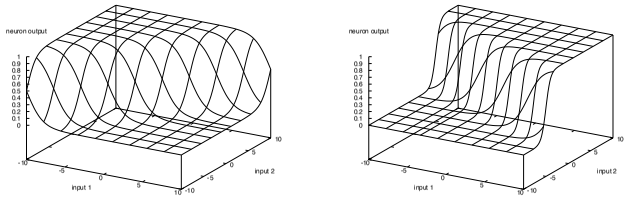
Output artificial neuron

$$\frac{1}{1+e^{-(1*in_1+1*in_2+0)}} \qquad \frac{1}{1+e^{-(1*in_1+1*in_2+5)}}$$



Output artificial neuron

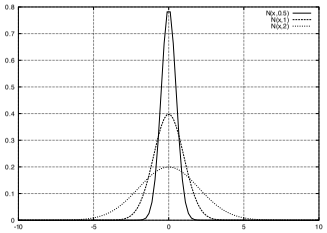
$$\frac{1}{1+e^{-(-1*in_1+1*in_2+0)}} \qquad \frac{1}{1+e^{-(1*in_1+2*in_2+0)}}$$



Evolutionary search for network weights

- Mutate neural network weights by adding with each weight a small random, Gaussian distributed number.
- Each weight has its own Gaussian distribution (different widths or variances).
- The width or variance of each Gaussian distribution also evolves by mutation (*technical details omitted here*).

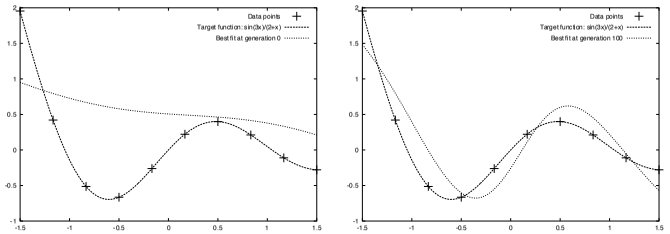
Normal distribution



Example: learning simple function

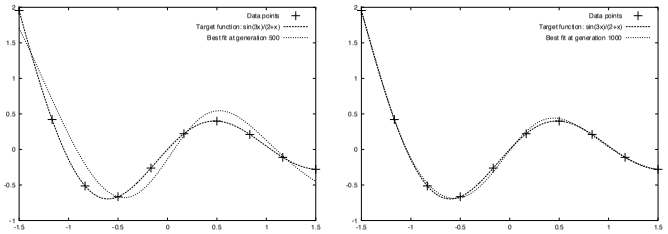
- Population size = 30
- Truncation selection (select best 50%)
- Only Gaussian mutation (1 mutation parameter for each weight)
- Target function: $\frac{\sin(3x)}{2}$, $x \in [-1.5 : 1.5]$
- 10 data points
- Neural network layers: 1 - 10 - 1
- hidden neurons: Tanh activation function
- Output neuron: linear activation function

EA regression



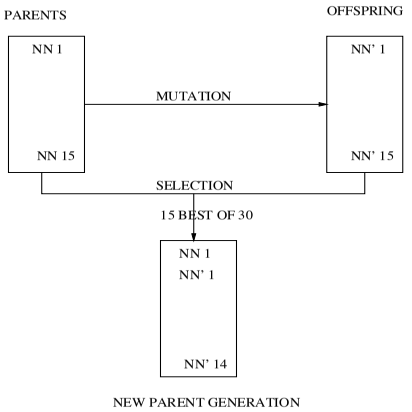
0 generations 100 generations

EA regression

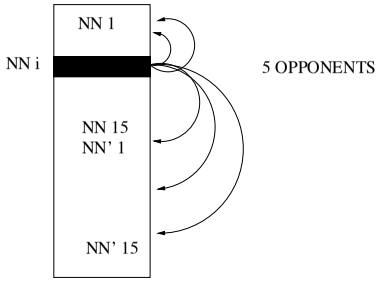


500 generations 1000 generations

Evolutionary cycle



Fitness evaluation



15 parents + 15 offspring neural networks
Each NN competes against 5 randomly chosen NN
Score: win : + 1; draw : 0; loss : -2
Fitness: sum of scores

Darwinian system ?

1. Structures ?
   → **neural networks**
2. Structures are copied ?
   → **15 parents ⇒ 15 offspring**
3. Copies partially vary from the original
   → **weights Gaussian mutated**
4. Structures are competing for a limited resource
   → **fixed population size each generation**
5. Reproductive success depends on environment
   → **winning strategies survive**

Experiment

- total of 250 generations evolved
- 15 neural networks each generation
  ⇒ 15 x 250 = 3750 neural networks created
- fitness evaluation: 15 parents + 15 offspring: each competing against 5 others
  ⇒ 30 x 5 x 250 = 37500 games played

Performance

90 games played on Internet checkers site (anonymous)

rating	level
2400+	senior master
2200 – 2399	master
2000 – 2199	expert
1800 – 1999	class A
1600 – 1799	class B
1400 – 1599	class C
1200 – 1399	class D

rating	win	draw	loss
2100+	1	1	5
2000 – 2099	2	0	6
1900 – 1999	3	1	5
1800 – 1899	5	3	6
1700 – 1799	10	5	3
1600 – 1699	19	5	0
1500 – 1599	7	0	0
1400 – 1499	2	0	0

- Evolved neural network rating: 1902
- 1 draw against player rated 2207 , ie. master level, ranked 18 out of 40000 listed players

Discussion

- Chinook: opening games, end games table look-up, hand-crafted evaluation function
- Deep Blue: 200 million board evaluations per second vs. 3500 here
- Payoff: summed score over 5 games → no immediate feedback about winning or losing a single game
- Input to neural networks do not give spatial information: only 1 x 32 vector of {-K,-1,0,1,K}

Newell: "It is extremely doubtful whether there is enough information in 'win, lose, or draw' when referred to the whole play of the game to permit any learning at all over available time scales"

Conclusion

- Building intelligent systems by evolutionary computing.
- Learn to play checkers at high level by competing against themselves.
- No tedious domain knowledge extraction.
- Learning by evolution: feasible approach.