Learning to Play Games
IEEE CIG 2008 Tutorial

Simon M. Lucas
University of Essex, UK

Aims

- To provide a practical guide to the main machine learning methods used to learn game strategy
- Provide insights into when each method is likely to work best
  - Details can mean difference between success and failure
- Common problems, and some solutions
- We assume you are familiar with
  - Neural networks: MLPs and Back-Propagation
  - Rudiments of evolutionary algorithms (evaluation, selection, reproduction/variation)
- Demonstrate TDL and Evolution in action

Overview

- Architecture (action selector v. value function)
- Learning algorithm (Evolution v. Temporal Difference Learning)
- Function approximation method
  - E.g. MLP or Table Function
  - Interpolated tables
- Information rates
- Sample games (Mountain Car, Othello, Ms. Pac-Man)

Architecture

- Where does the computational intelligence fit in to a game playing agent?
- Two main choices
  - Value function
  - Action selector
- First, let’s see how this works in a simple grid world

Action Selector

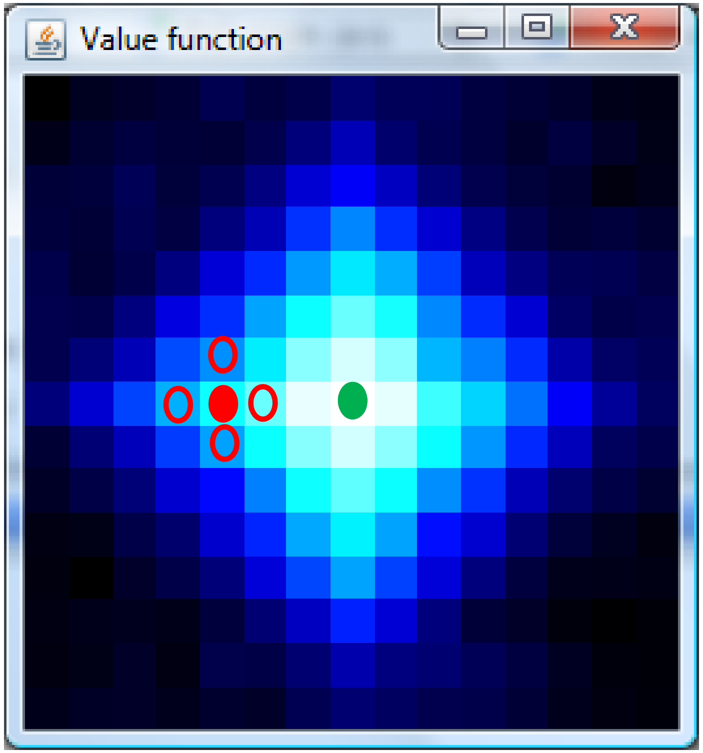
- Maps observed current game state to desired action
- For
  - No need for internal game model
  - Fast operation when trained
- Against
  - More training iterations needed (more parameters to set)
  - May need filtering to produce legal actions
  - Separate actuators may need to be coordinated

State Value Function

- Hypothetically apply possible actions to current state to generate set of possible next states
- Evaluate these using value function
- Pick the action that leads to the most favourable state
- For
  - Easy to apply, learns *relatively* quickly
- Against
  - Need a model of the system

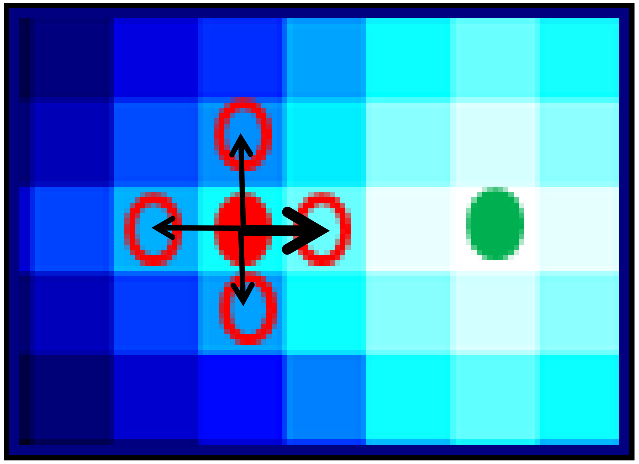
Grid World

- n x n grid (toroidal i.e. wrap-around)
- Reward: 0 at goal, -1 elsewhere
- State: current square {i, j}
- Actions: up, down, left, right
- Red Disc: current state
- Red circles: possible next states
- Each episode: start at random place on grid and take actions according to policy until the goal is reached, or maximum iterations have been reached
- Examples below use 15 x 15 grid

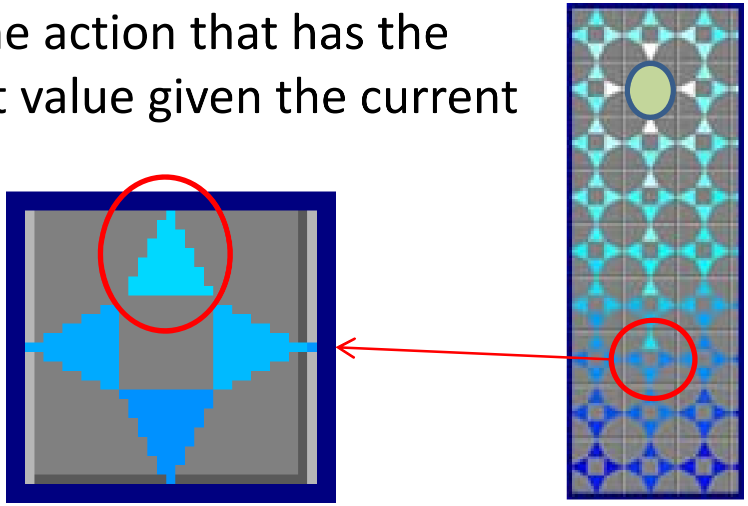


State Value versus State-Action Value: Grid World Example

- State value: consider the four states reachable from the current state by the set of possible actions
  - choose action that leads to highest value state



- State-Action Value
  - Take the action that has the highest value given the current state



Run Demo:
Time to see each approach in action

Learning Algorithm: (Co) Evolution v. TDL

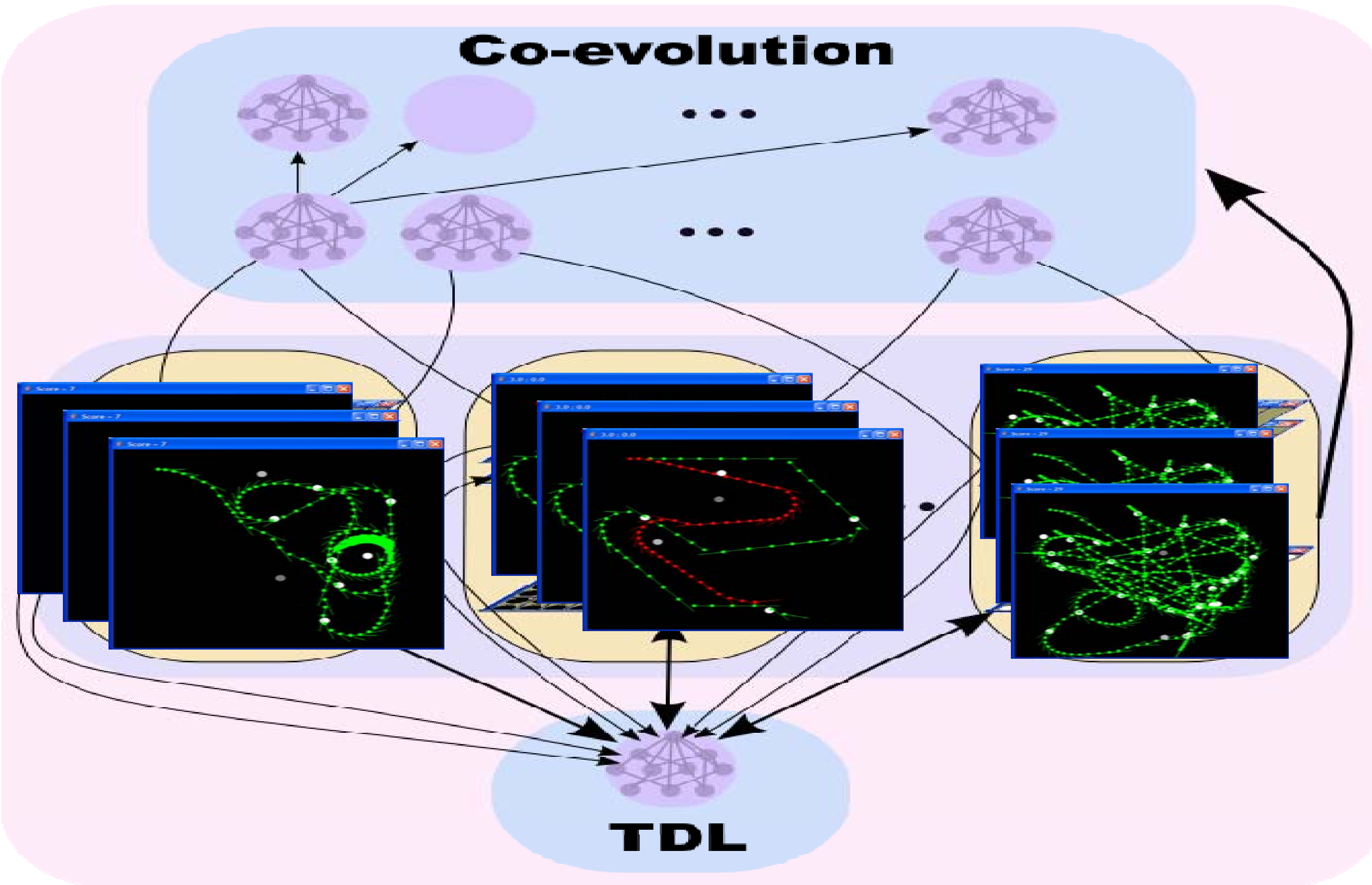
- Temporal Difference Learning
  - Often learns much faster
  - But less robust
  - Learns during game-play
  - Uses information readily available (i.e. current observable game-state)
- Evolution / Co-evolution (vanilla form)
  - Information from game result(s)
  - Easier to apply
  - But wasteful: discards *so much* information
- Both can learn game strategy from scratch

Co-evolution (single population)

Evolutionary algorithm: rank them using a league

	Team	P	W	D	L	F	A	GD	PTS
1	Arsenal	6	5	1	0	15	4	11	16
2	Man Utd	7	4	2	1	6	2	4	14
3	Manchester City	7	4	1	2	8	5	3	13
4	Liverpool	6	3	3	0	11	2	9	12
5	Newcastle	6	3	2	1	9	5	4	11
6	Chelsea	7	3	2	2	7	8	-1	11
7	West Ham	6	3	1	2	9	6	3	10
8	Aston Villa	6	3	1	2	7	4	3	10
9	Everton	7	3	1	3	8	8	0	10
10	Blackburn	6	2	3	1	5	4	1	9
11	Portsmouth	7	2	3	2	8	8	0	9
12	Wigan	7	2	2	3	8	7	1	8
13	Middlesbrough	7	2	2	3	9	11	-2	8
14	Birmingham	7	2	2	3	7	9	-2	8
15	Sunderland	7	2	2	3	7	11	-4	8
16	Reading	7	2	1	4	5	11	-6	7
17	Fulham	7	1	3	3	12	14	-2	6
18	Tottenham	7	1	2	4	10	12	-2	5
19	Bolton	7	1	1	5	8	12	-4	4
20	Derby	7	1	1	5	4	20	-16	4

In Pictures…



Information Flow

- Interesting to observe information flow
- Simulating games can be expensive
- Want to make the most of that computational effort
- Interesting to consider bounds on information gained per episode (e.g. per game)
- Consider upper bounds
  - All events considered equiprobable

Evolution

- Suppose we run a co-evolution league with 30 players in a round robin league (each playing home and away)
- Need n(n-1) games
- Single parent: pick one from n
- log_2(n)
- Information rate: $$I_c = \frac{\log_2 n}{n(n-1)}$$

n	$I_c(\mathrm{bg}^{-1})$
2	0.500
5	0.12
10	0.037
30	0.006

TDL

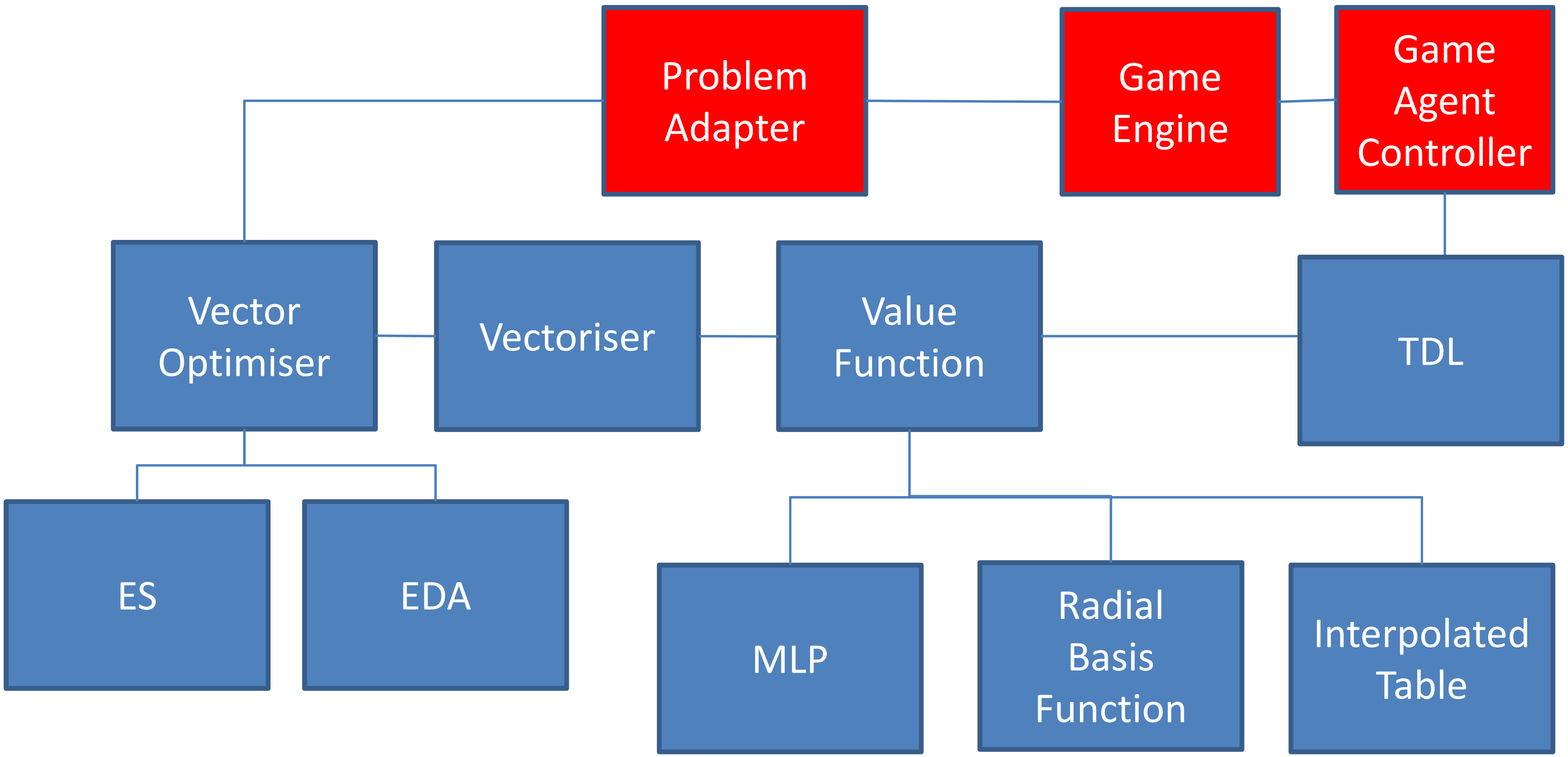
- Information is fed back as follows:
  - 1.6 bits at end of game (win/lose/draw)
- In Othello, 60 moves
- Average branching factor of 7
  - 2.8 bits of information per move
  - 60 * 2.8 = 168
- Therefore:
  - Up to nearly 170 bits per game (> 20,000 times more than coevolution for this scenario)
  - (this bound is very loose – why?)
- See my CIG 2008 paper

Sample TDL Algorithm: TD(0)
typical alpha: 0.1
pi: policy; choose rand move 10% of time
else choose best state

Algorithm 1: On-line TD(0) adapted from Sutton and Barto

INITIALIZE $V(s)$ arbitrarily, for all $s \in S$
for *each episode* **do**
  Initialize s to start state
  (could be random start state)
  for *each step in episode* **do**
    $a \leftarrow$ action given by π for s
    Take action a, observe reward r, and next state s'
    $\delta \leftarrow r + V(s') - V(s)$
    $V(s) \leftarrow V(s) + \alpha\delta$
  end
end

Main Software Modules
(my setup – plug in game of choice)



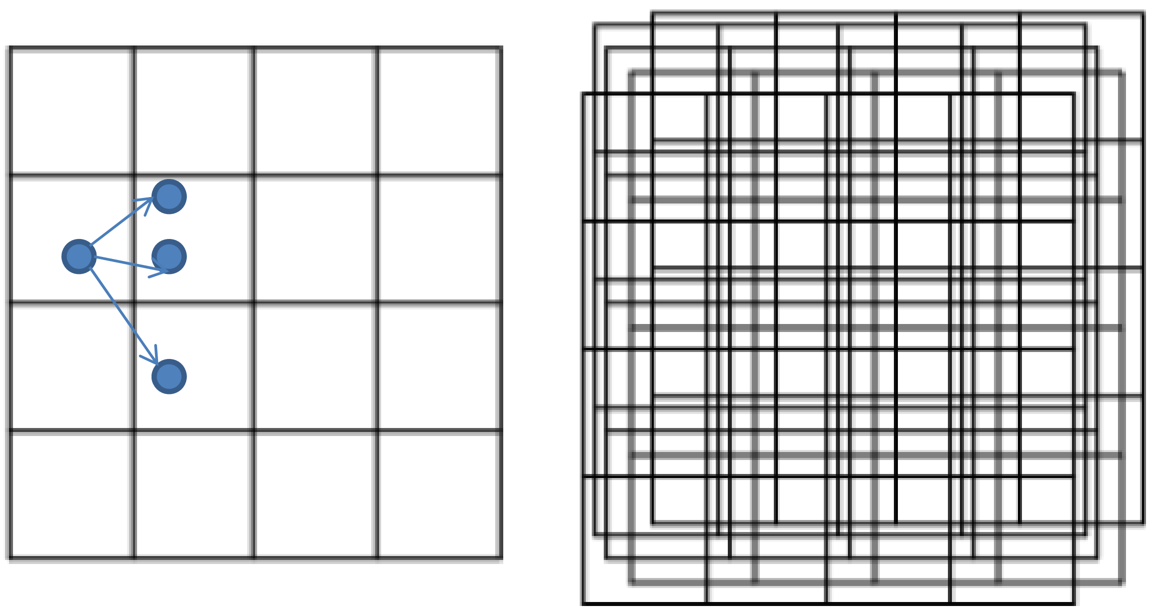
Function Approximators

- For small games (e.g. OXO) game state is so small that state values can be stored directly in a table
- Our focus is on more complex games, where this is simply not possible e.g.
  - Discrete but large (Chess, Go, **Othello, Pac-Man**)
  - Continuous (**Mountain Car**, Halo, Car racing: **TORCS**)
- Therefore necessary to use a function approximation technique

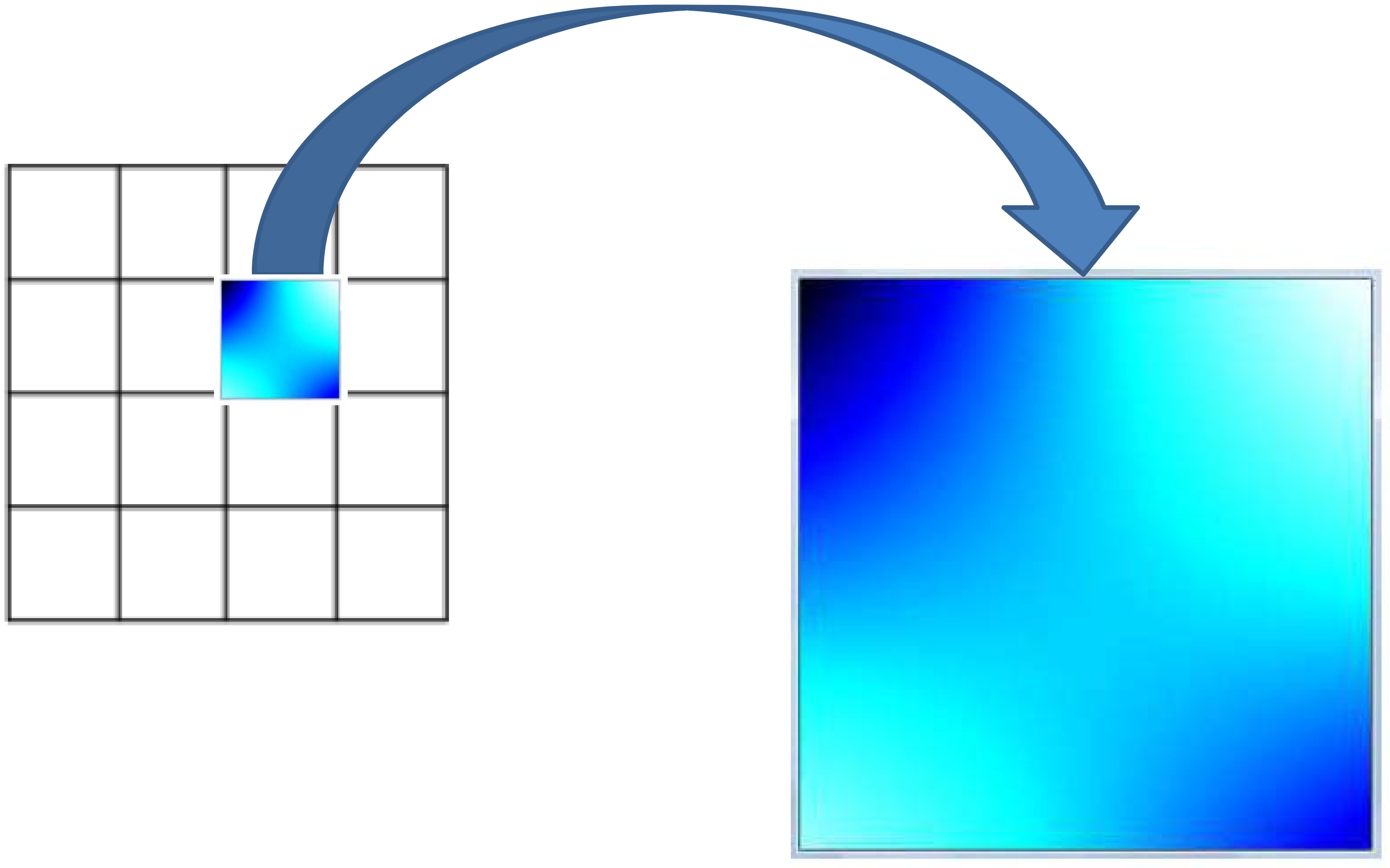
Function Approximators

- Multi-Layer Perceptrons (MLPs)
  - Very general
  - Can cope with high-dimensional input
  - Global nature can make forgetting a problem
- N-Tuple systems
  - Good for discrete inputs (e.g. board games)
  - Harder to apply to continuous domains
- Table-based
  - Naïve is poor for continuous domains
  - CMAC coding improves this (overlapping tiles)
  - Even better: use interpolated tables
    - Generalisation of bilinear interpolation used in image transforms

Standard (left) versus CMAC (right)



Interpolated Table



Method

- Continuous point p(x,y)
- x and y are discretised, then residues r(x) r(y) are used to interpolate between values at four corner points
- N-dimensional table requires 2^n lookups

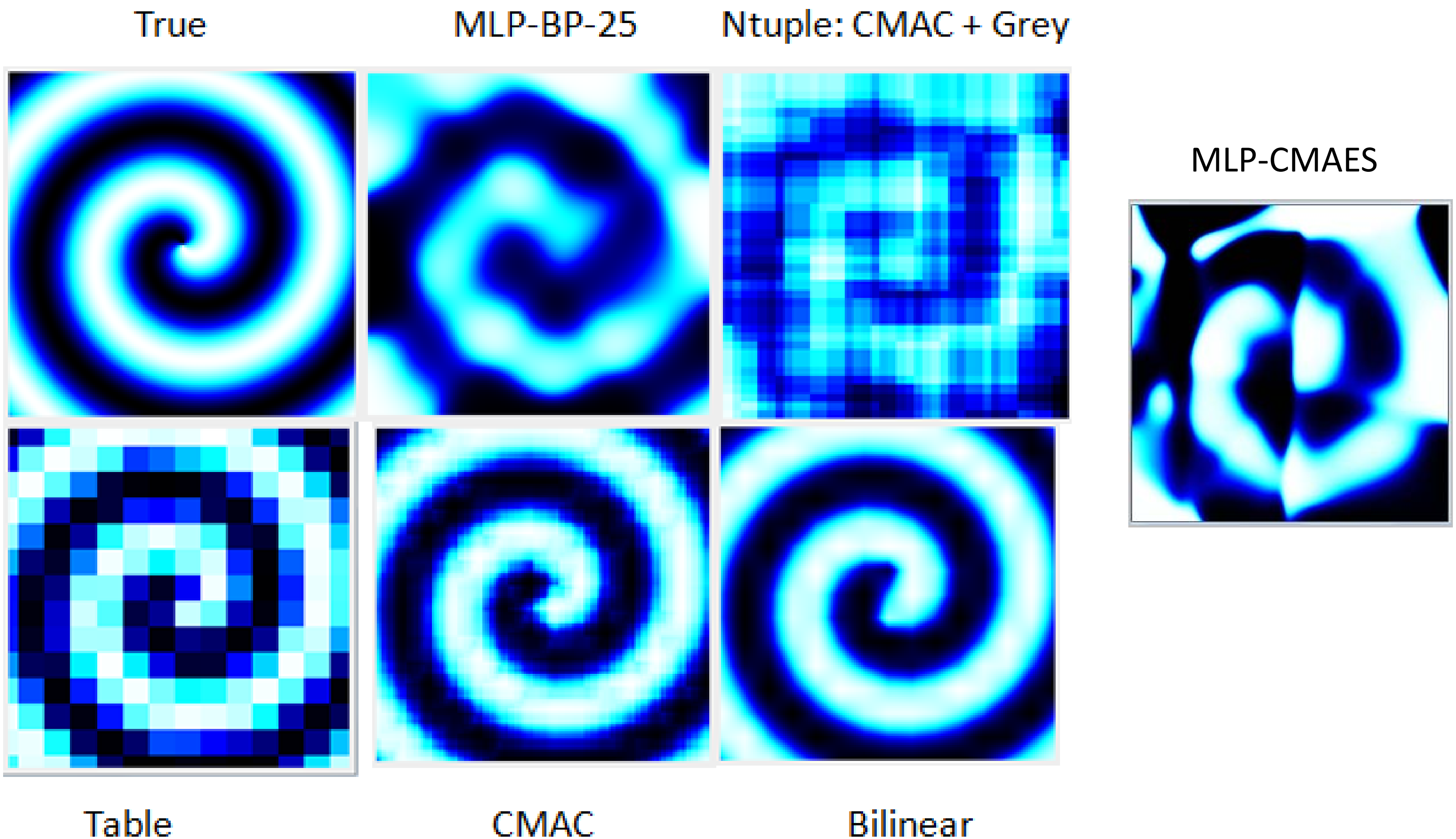
$$
\begin{aligned}
f_t(x,y) \quad &= \quad (1-r(x))(1-r(y))t[q_l(x)][q_l(y)] \\
&+ \quad r(x)(1-r(y))t[q_u(x)][q_l(y)] \\
&+ \quad (1-r(x))r(y)t[q_l(x)][q_u(y)] \\
&+ \quad r(x)r(y)t[q_u(x)][q_u(y)]
\end{aligned}
$$

Supervised Training Test

- Following based on 50,000 one-shot training samples
- Each point randomly chosen from uniform distribution over input space
- Function to learn: continuous spiral (r and theta are the polar coordinates of x and y)

$$f(x, y) = \sin(\theta + r\pi\omega)$$

Results

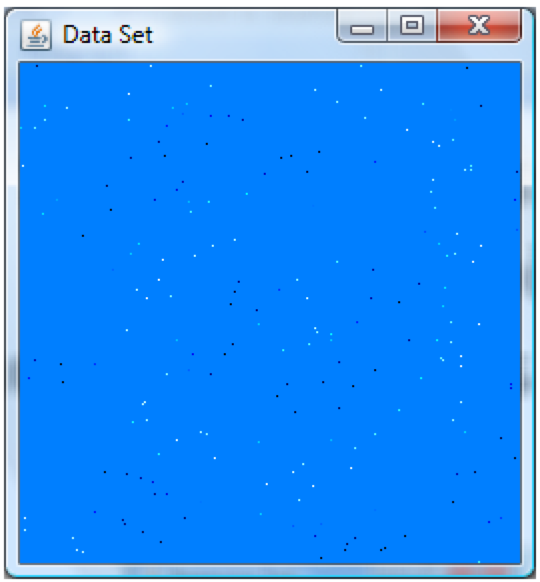

True
MLP-BP-25
Ntuple: CMAC + Grey
MLP-CMAES
Table
CMAC
Bilinear

Test Set MSE

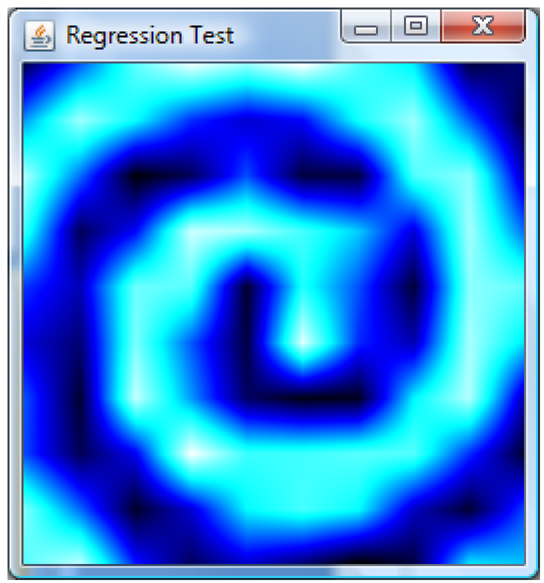
Architecture	MSE
MLP	0.13
N-Tuple (CMAC + Grey)	0.30
Standard Table	0.08
CMAC (Shared)	0.01
Bi-Linear	0.006

Standard Regression
200 Training Points
Gaussian Processes Model

Data set



Interpolated Table

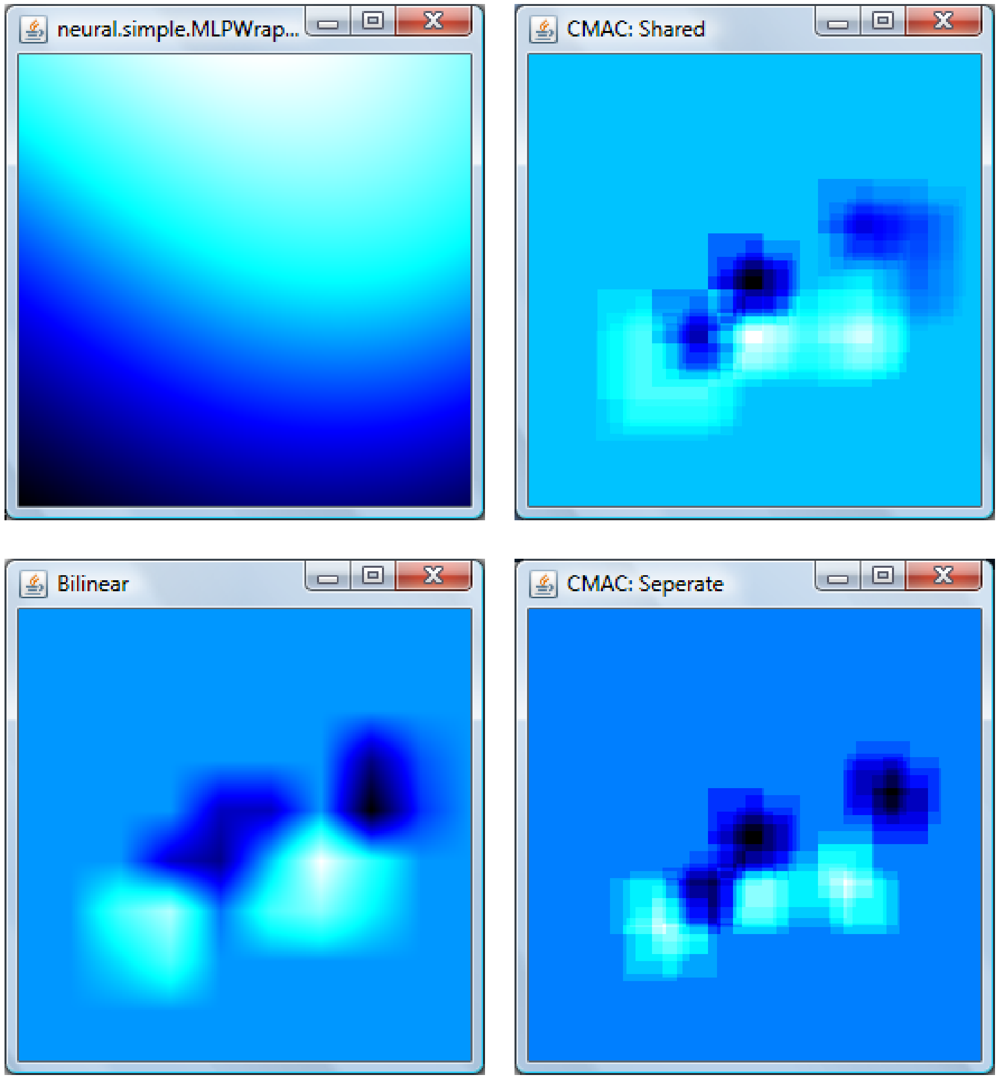


Gaussian Processes

Gaussian Processes: learn more from the data, but hard to interface to games

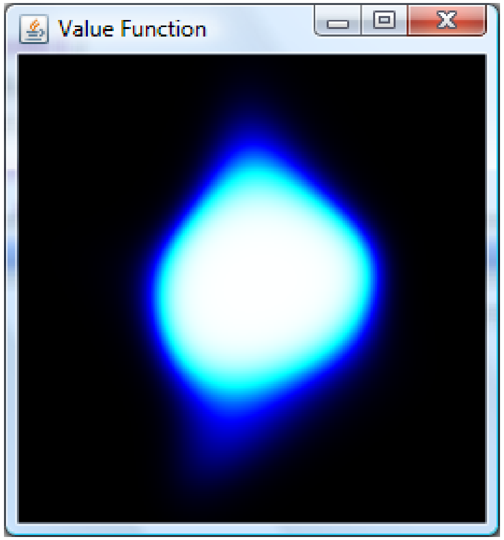
Function Approximator: Adaptation Demo

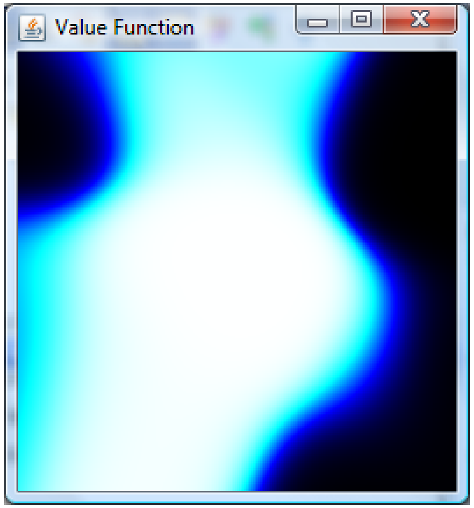
This shows each method after a single presentation of each of six patterns, three positive, three negative. What do you notice?



Grid World – Evolved MLP

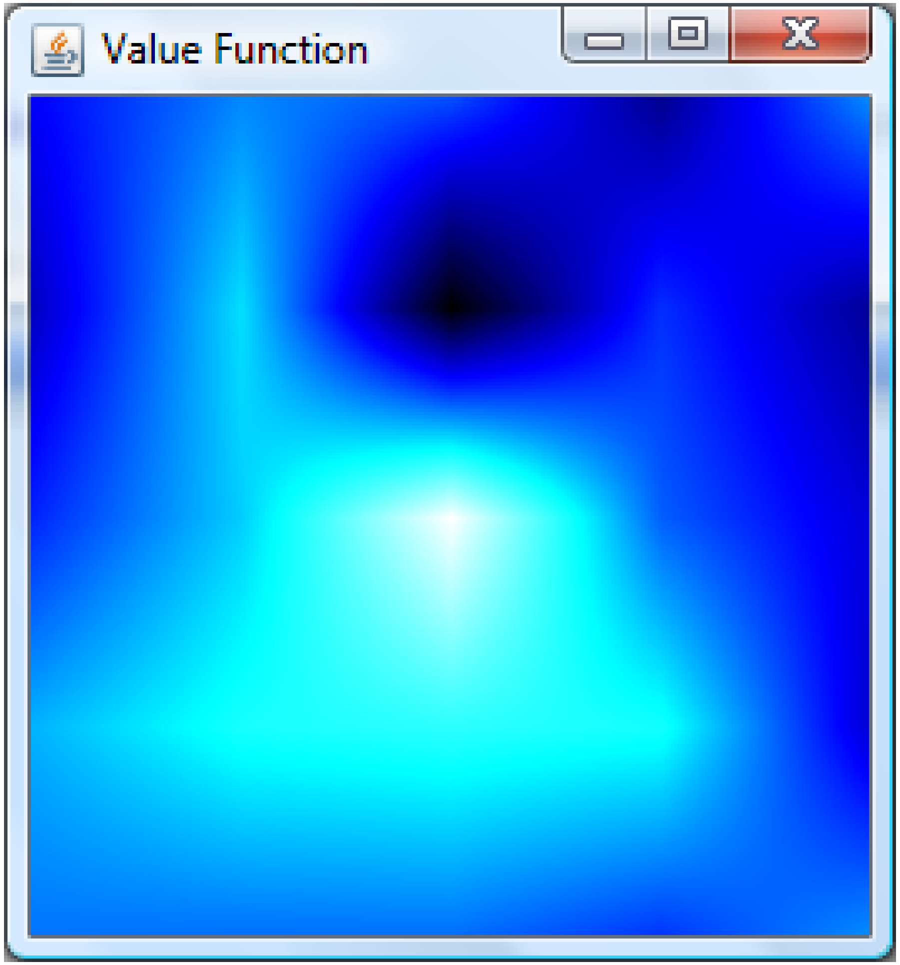
- MLP evolved using CMA-ES
- Gets close to optimal after a few thousand fitness evaluations
- Each one based on 10 or 20 episodes
- Value functions may differ from run to run





Evolved N-Linear Table

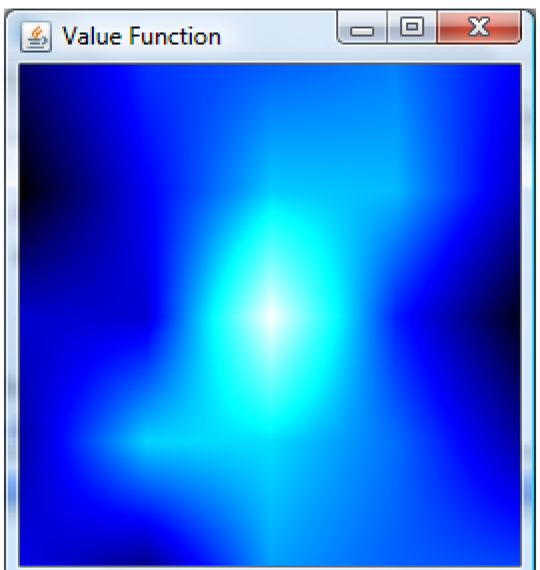
- This was evolved using CMA-ES, but only had a fitness of around 80

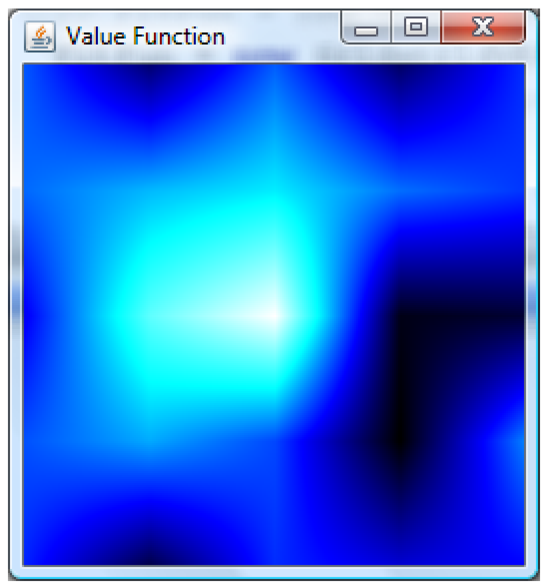


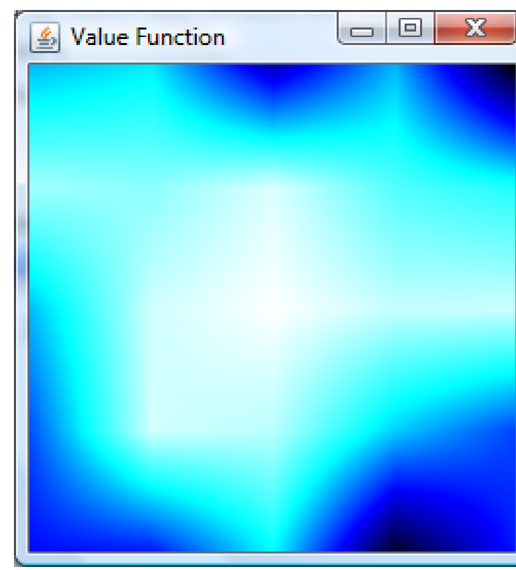
Evolved N-Linear Table with Lamarkian TD-Learning

- This does better
- Average score now 8.4

Evo N-Linear 5

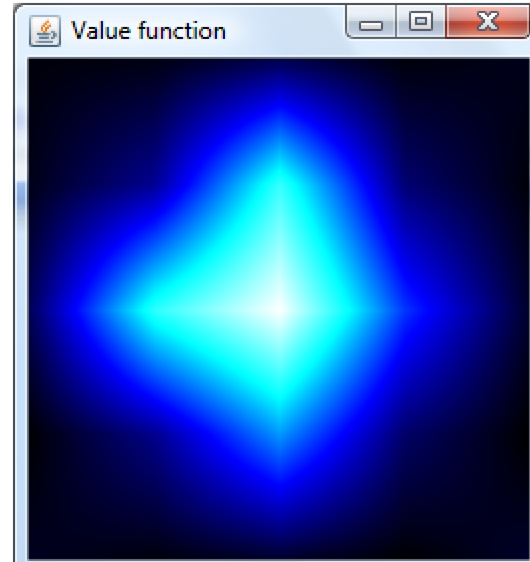


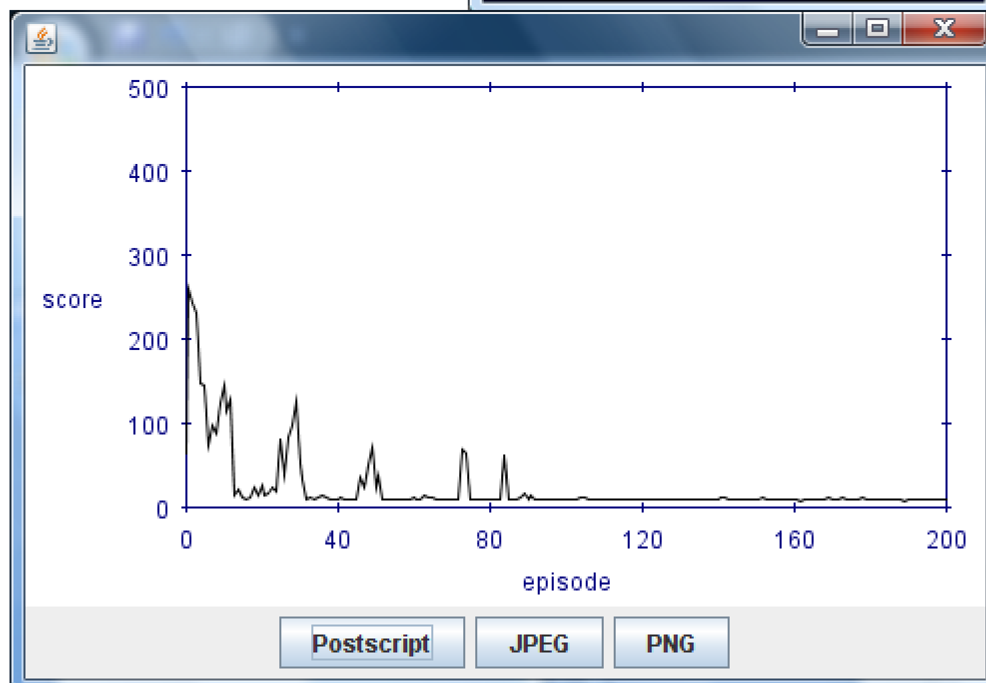




TDL Again

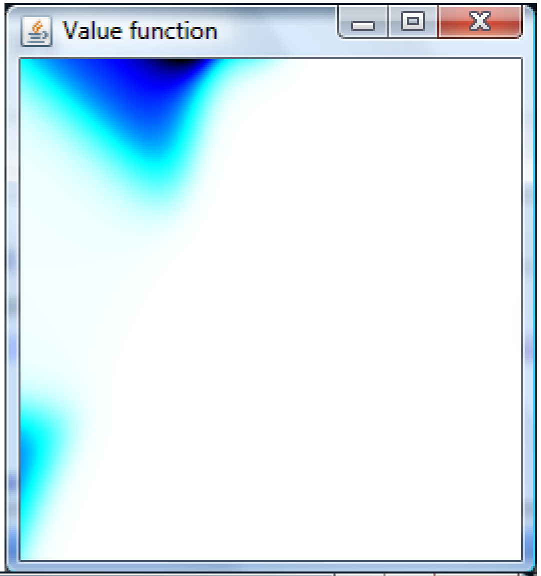
- Note how quickly it converges with the small grid





TDL MLP

- Surprisingly hard to make it work!



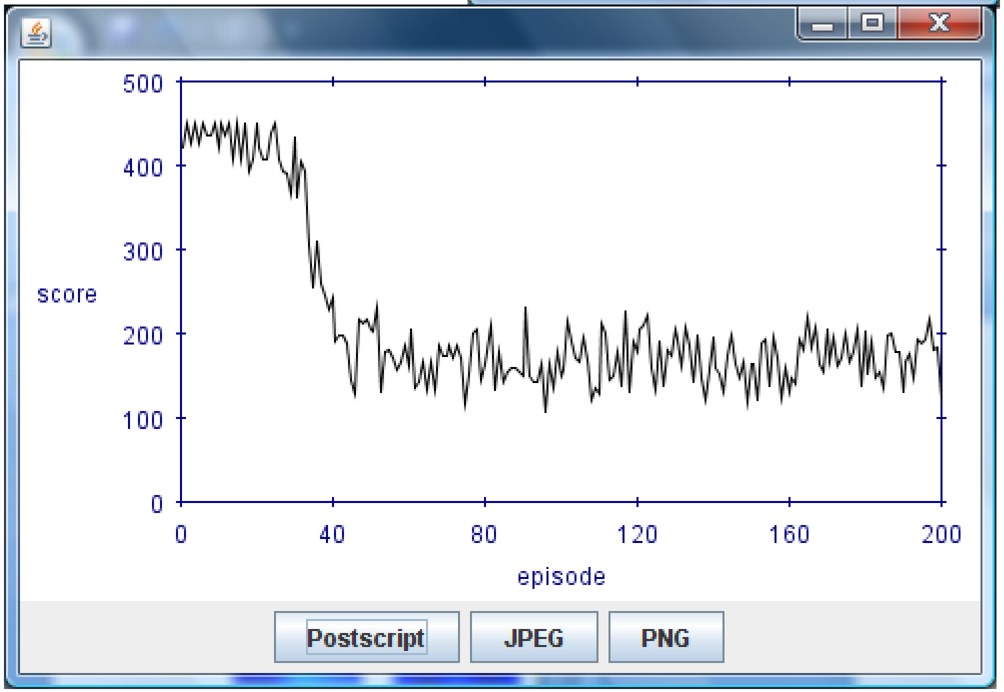
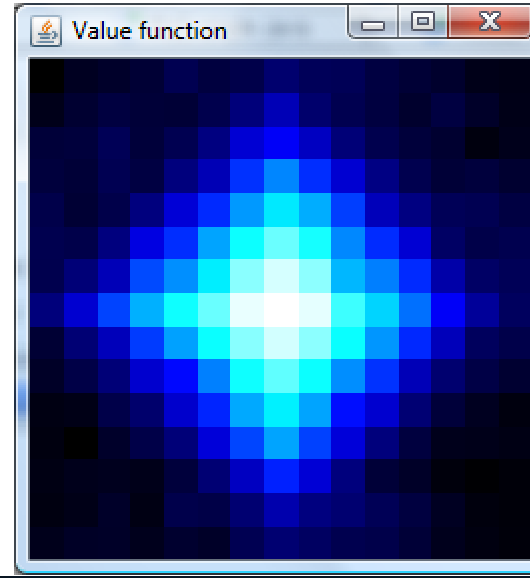
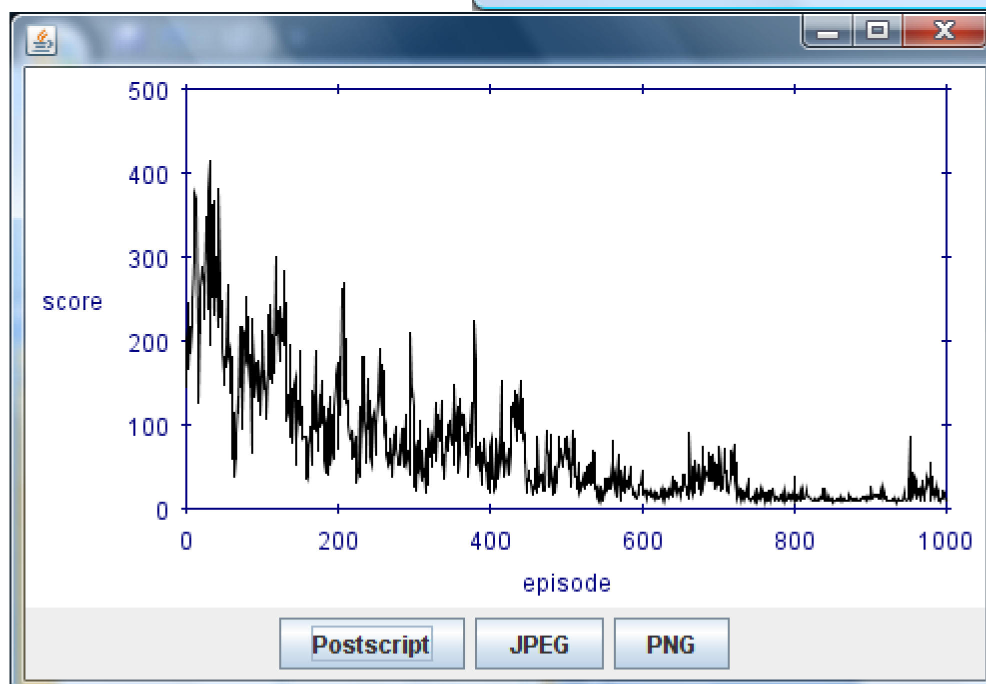


Table Function TDL (15 x 15)

- Typical score of 11.0
- Not as good as interpolated 5 x 5 table on this task
- Model selection is important



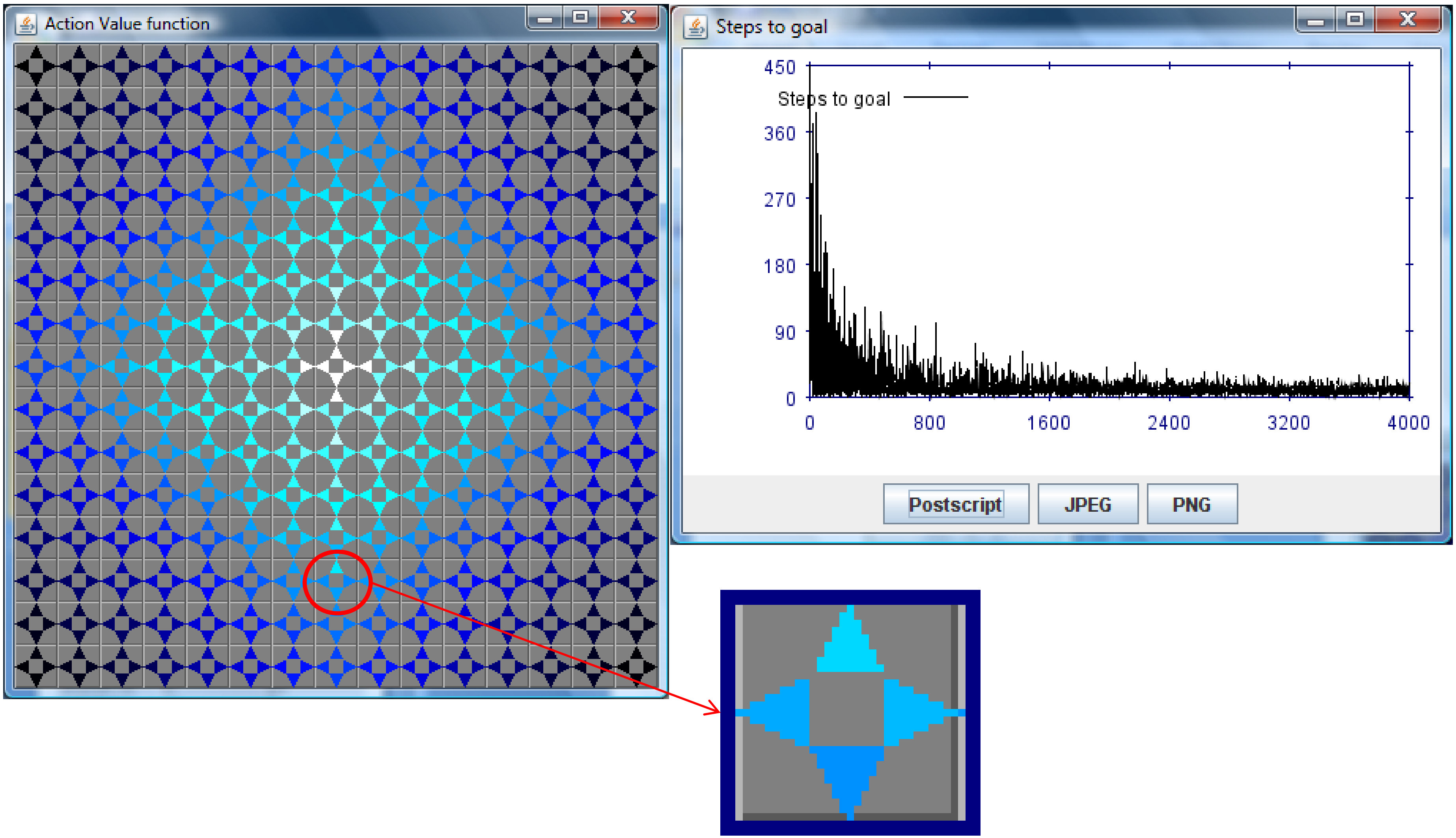


Grid World Results – State Table

- Interesting!
- The MLP / TDL combination is very poor
- Evolution with MLP gets close to TDL with N-Linear table, but at much greater computational cost

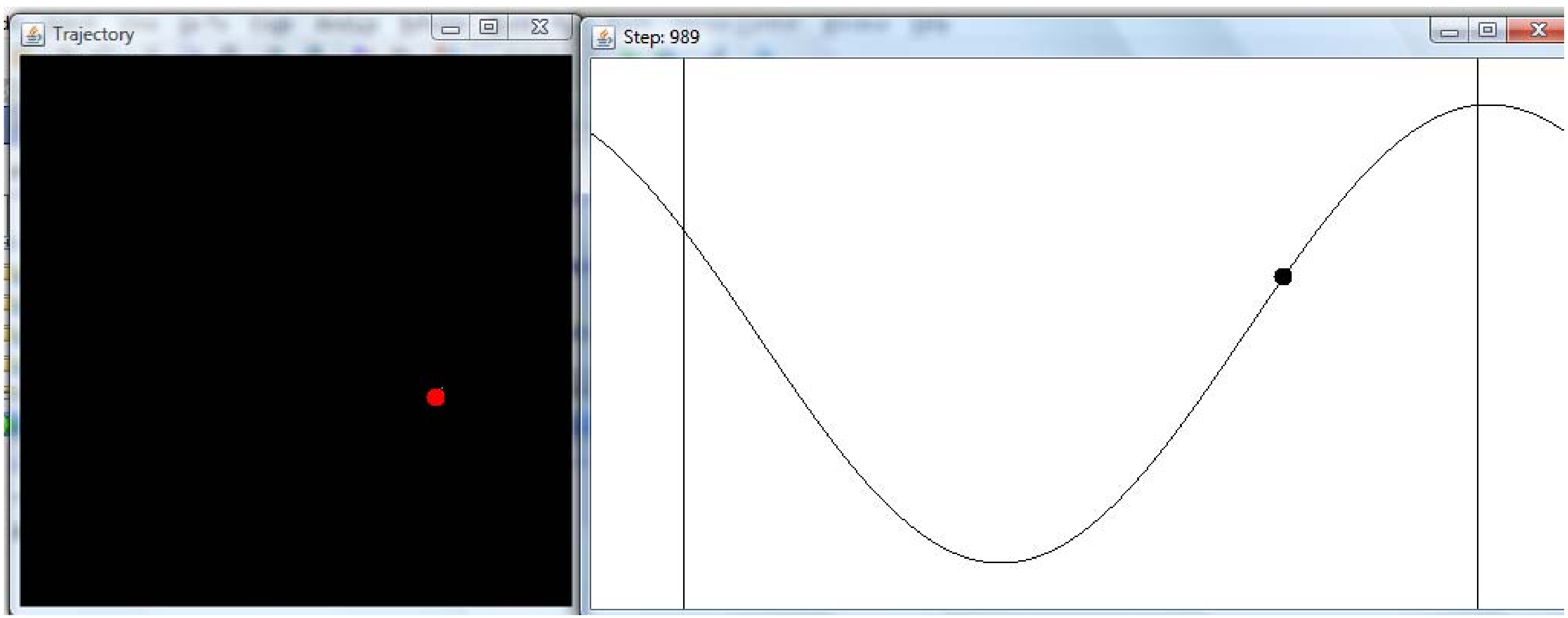
Architecture	Evolution (CMA-ES)	TDL(0)
MLP (15 hidden units)	9.0	126.0
N-Linear Table (5 x 5)	11.0	8.4

Action Values - Takes longer
e.g. score of 9.8 after 4,000 episodes

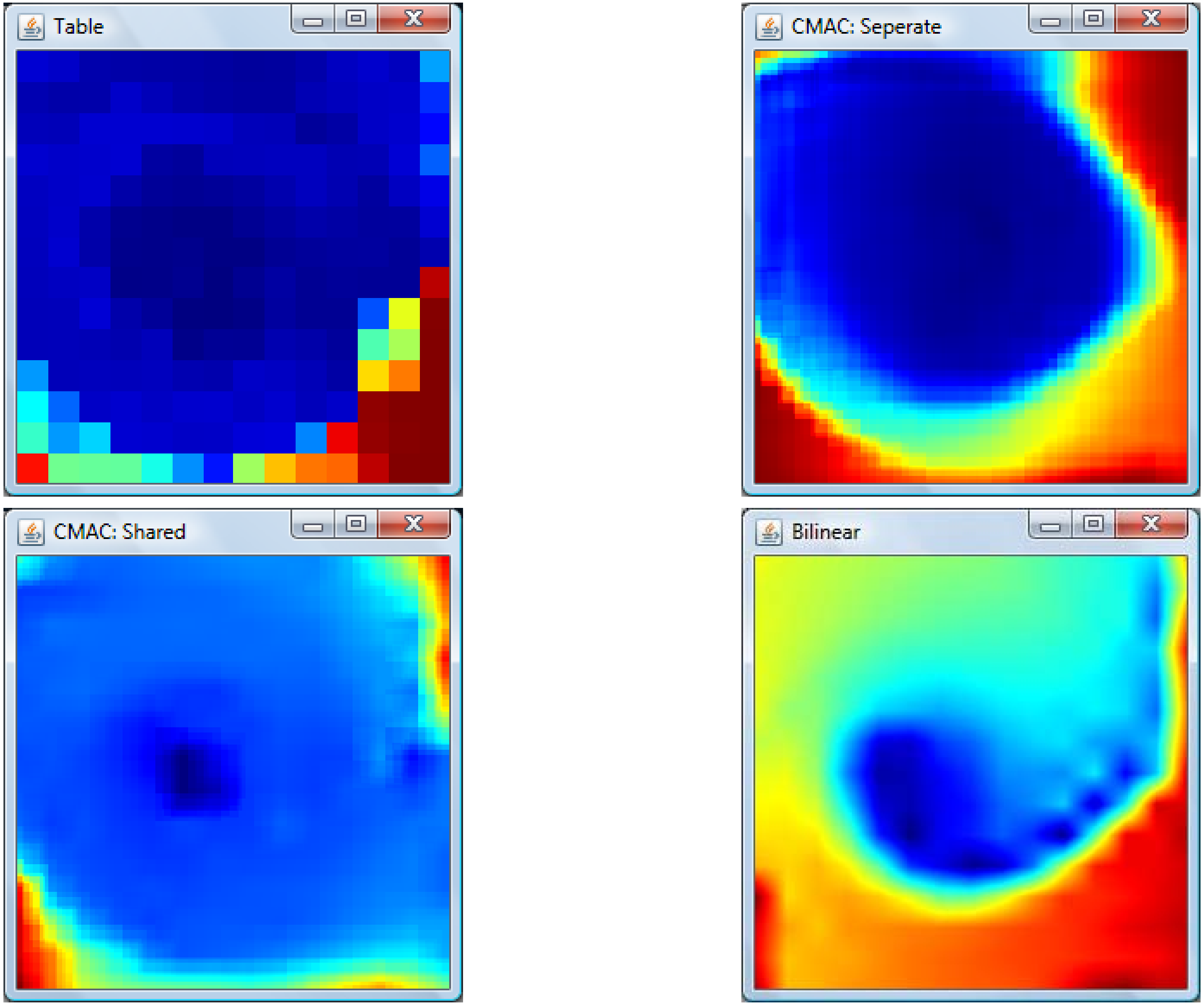


Simple Example: Mountain Car

- Standard reinforcement learning benchmark
- Accelerate a car to reach goal at top of incline
- Engine force weaker than gravity



Value Functions Learned (TDL)

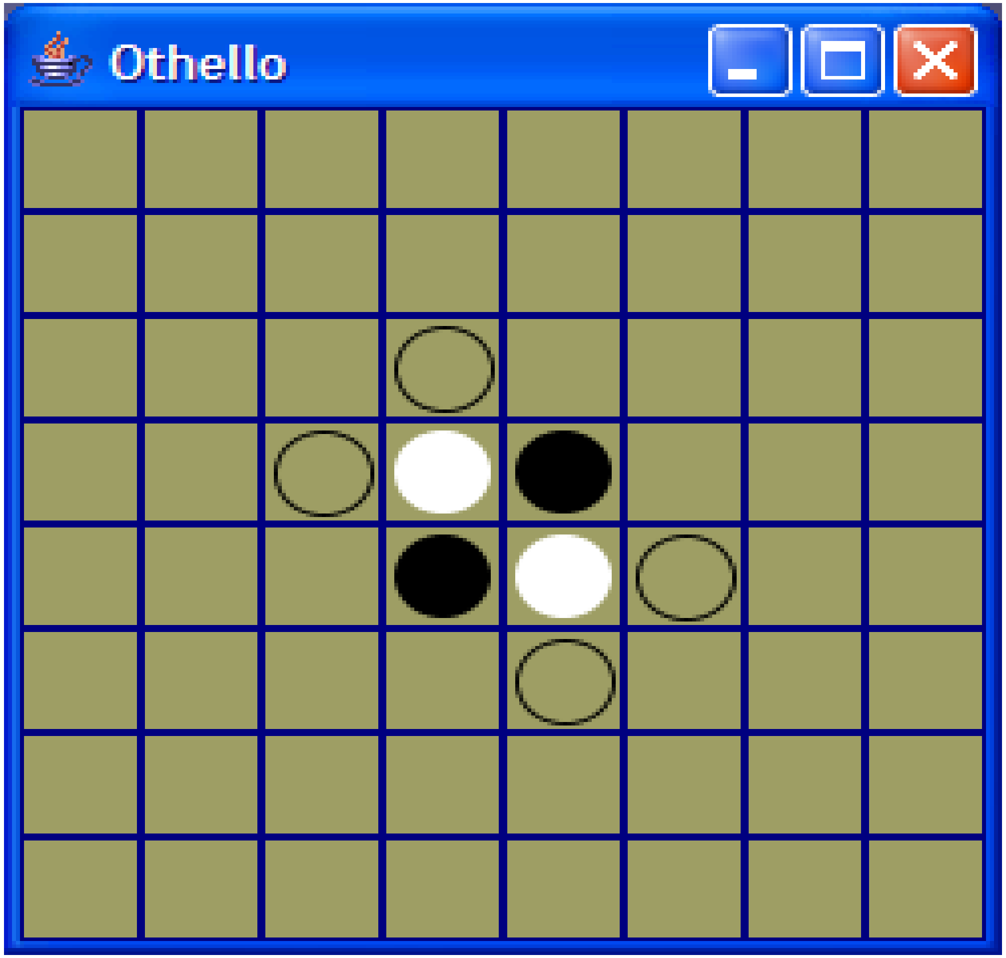


Mountain Car Results
(TDL, 2000 episodes, ave. of 10 runs)

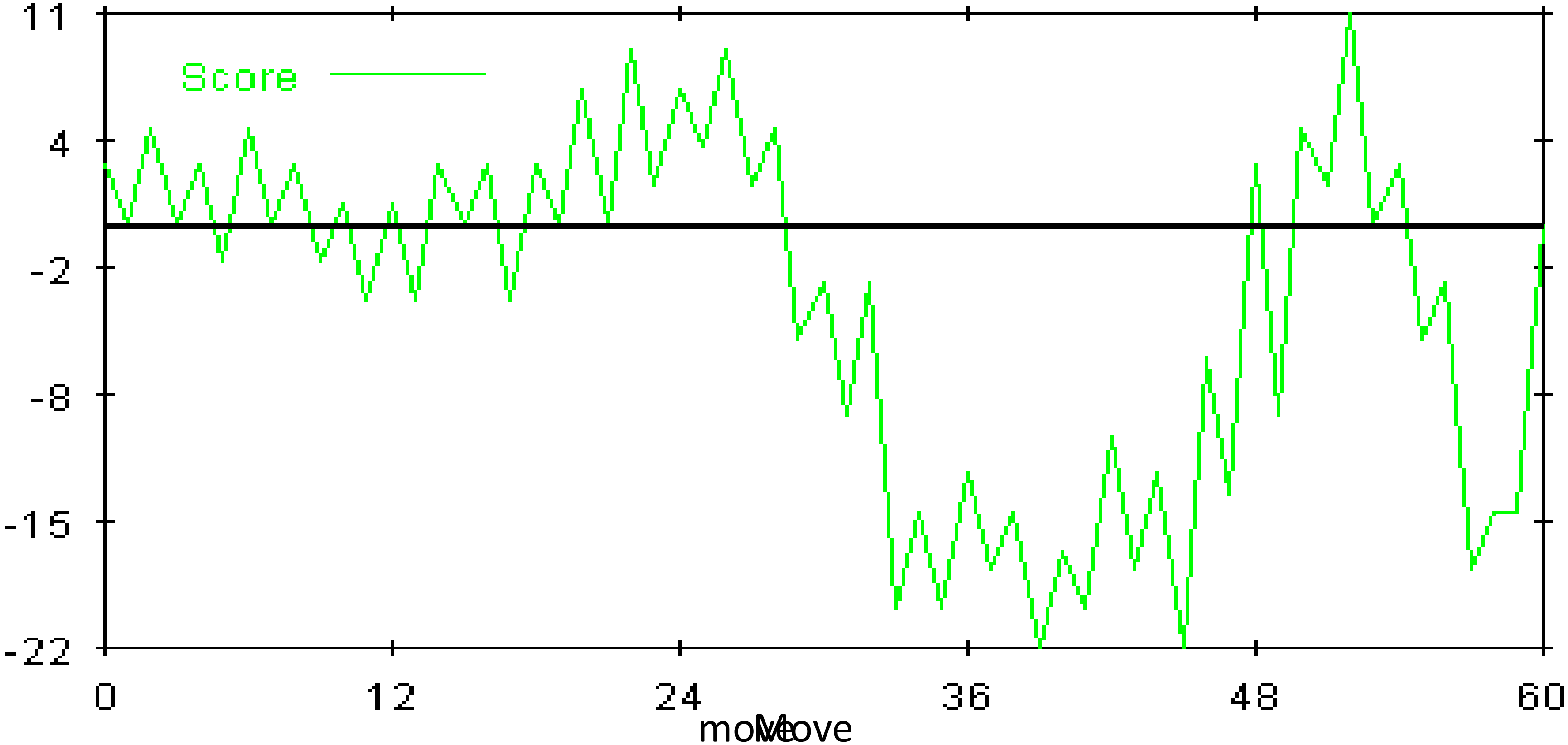
System	Mean steps to goal (s.e.)
Table	1008 (143)
CMAC: separate	81.8 (11.5)
CMAC: shared	60.0 (2.3)
Bilinear	**50.5 (2.5)**

Othello

See
Demo



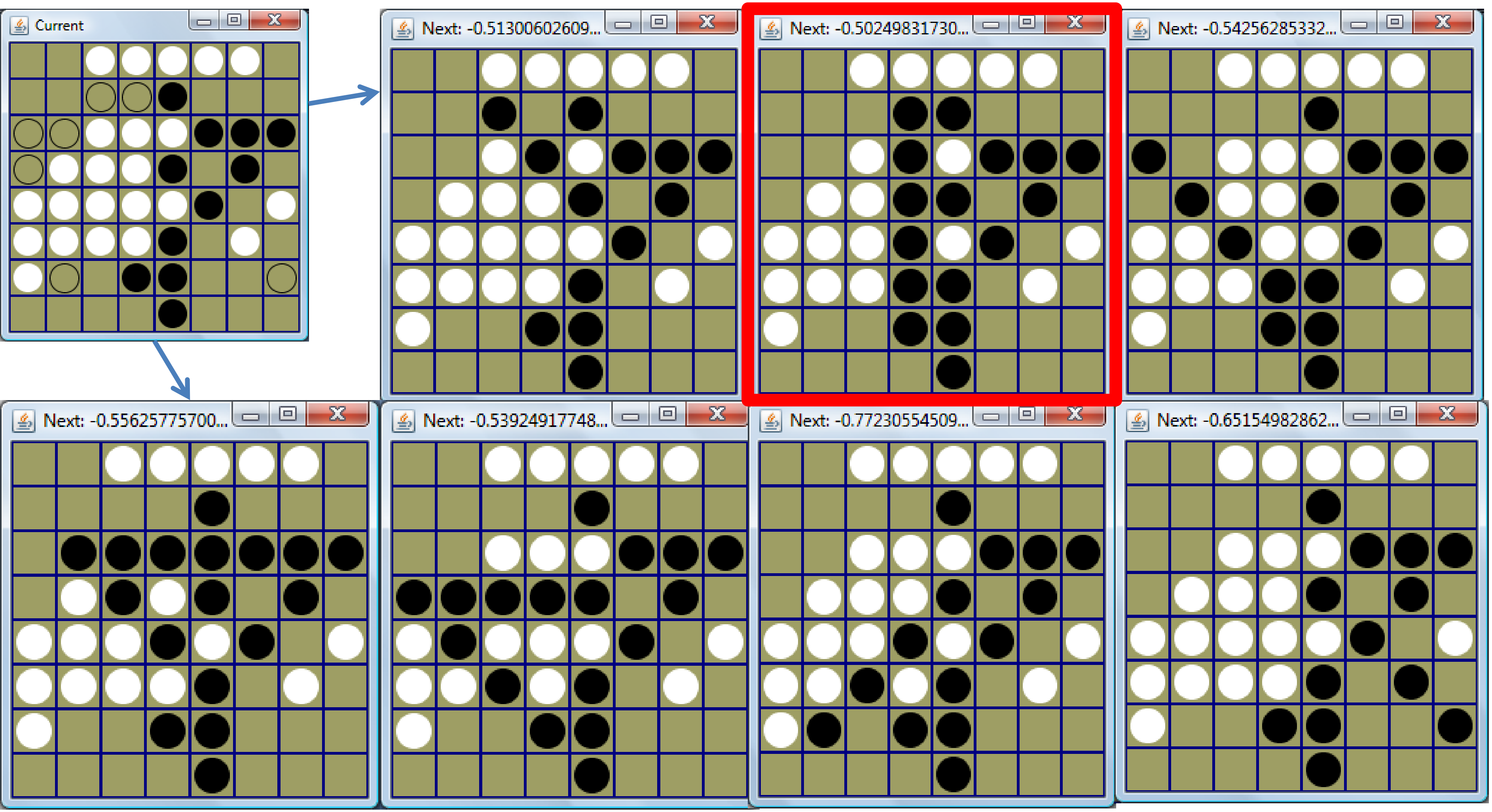
Volatile Piece Difference



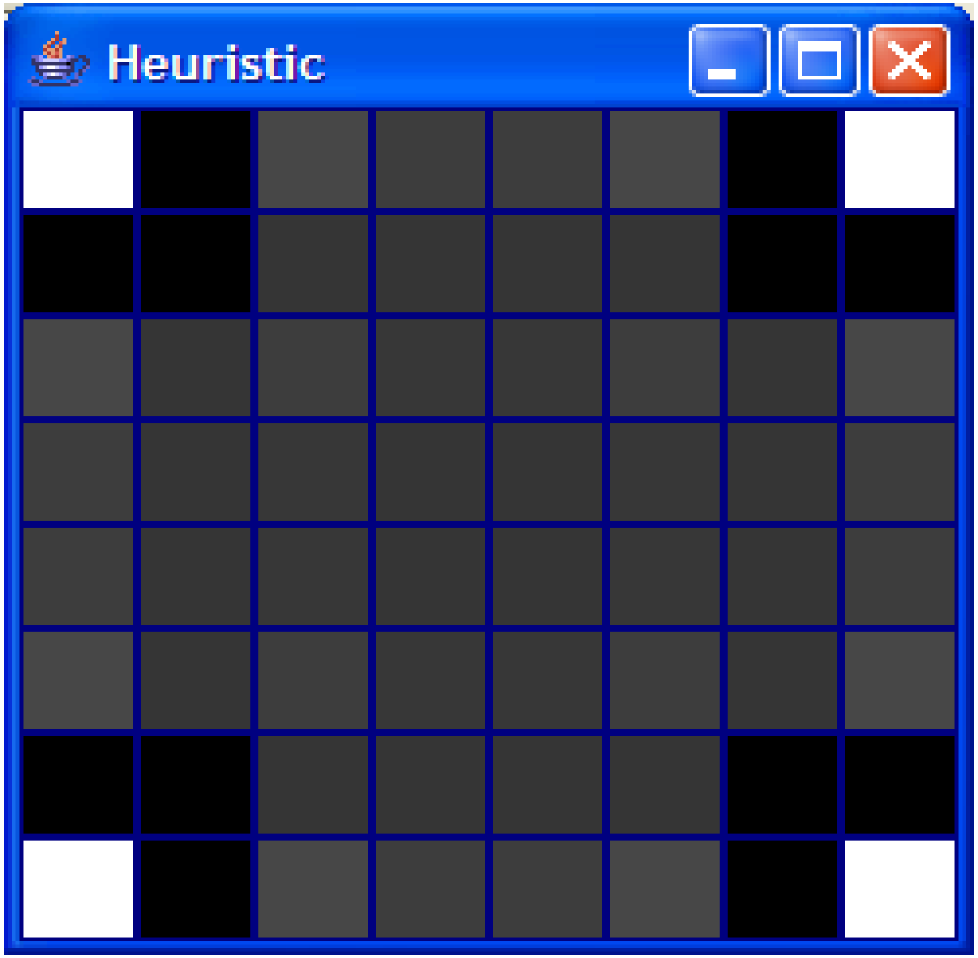
Setup

- Use weighted piece counter
  - Fast to compute (can play billions of games)
  - Easy to visualise
  - See if we can beat the ‘standard’ weights
- Limit search depth to 1-ply
  - Enables billions of games to be played
  - For a thorough comparison
- Focus on machine learning rather than game-tree search
- Force random moves (with prob. 0.1)
  - Get a more robust evaluation of playing ability

Othello: After-state Value Function



Standard “Heuristic” Weights (lighter = more advantageous)



TDL Algorithm

- Nearly as simple to apply as CEL

```
public interface TDLPlayer extends Player {
    void inGameUpdate(double[] prev, double[] next);
```

$$\alpha\left[v(x') - v(x)\right]\left(1 - v(x)^2\right)x_i$$

```
    void terminalUpdate(double[] prev, double tg);

}
```

$$\alpha\left[r - v(x)\right]\left(1 - v(x)^2\right)x_i$$

- Reward signal only given at game end
- Initial alpha and alpha cooling rate tuned empirically

TDL in Java

```
public void inGameUpdate(double[] prev, double[] next) {
    double op = tanh(net.forward(prev));
    double tg = tanh(net.forward(next));
    double delta = alpha * (tg - op) * (1 - op * op);
    net.updateWeights(prev, delta);
}

public void terminalUpdate(double[] prev, double tg) {
    double op = tanh(net.forward(prev));
    double delta = alpha * (tg - op) * (1 - op * op);
    net.updateWeights(prev, delta);
}
```

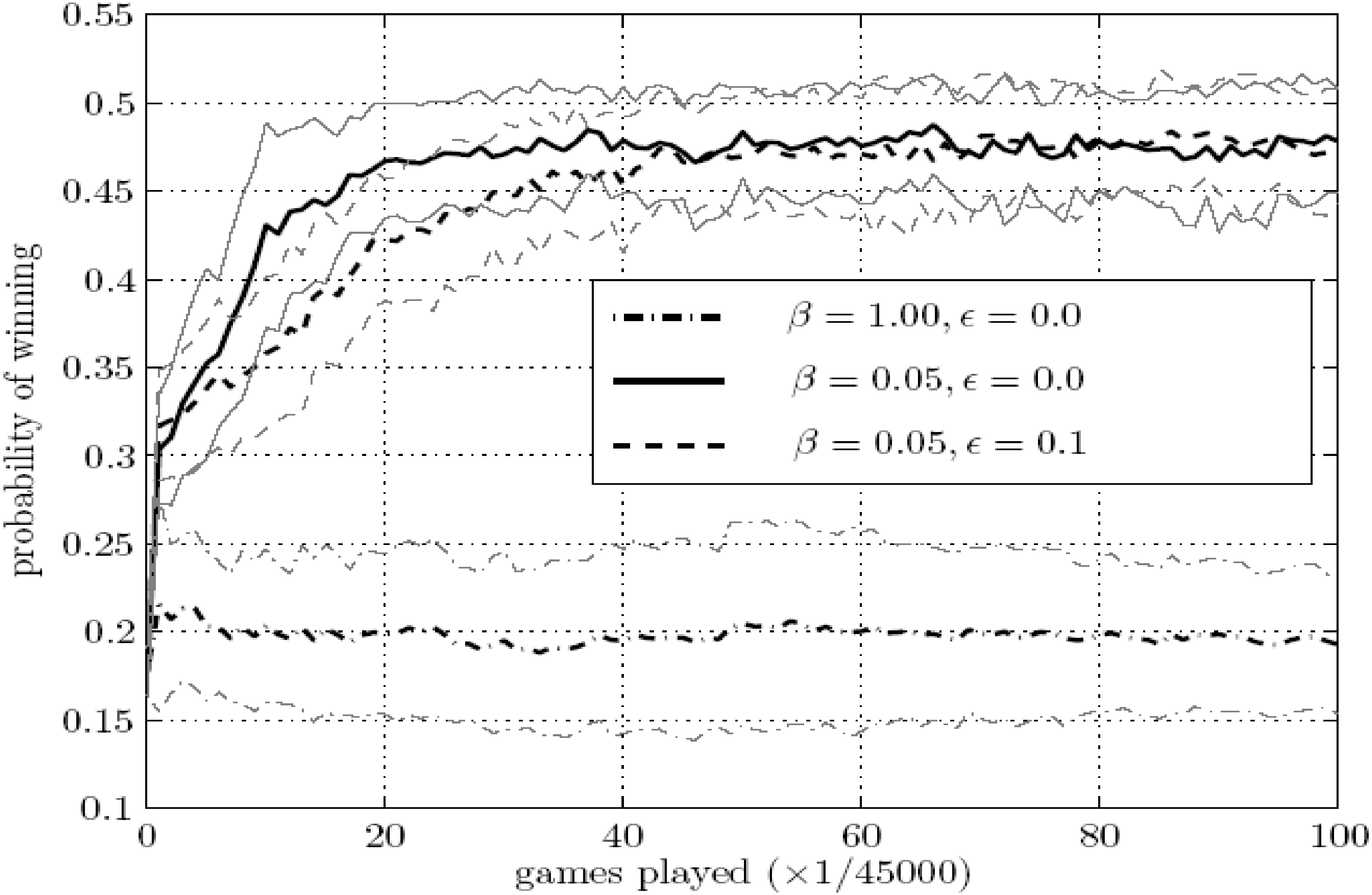
CEL Algorithm

- Evolution Strategy (ES)
  - (1, 10) (non-elitist worked best)
- Gaussian mutation
  - Fixed sigma (not adaptive)
  - Fixed works just as well here
- Fitness defined by full round-robin league performance (e.g. 1, 0, -1 for w/d/l)
- Parent child averaging
  - Defeats noise inherent in fitness evaluation

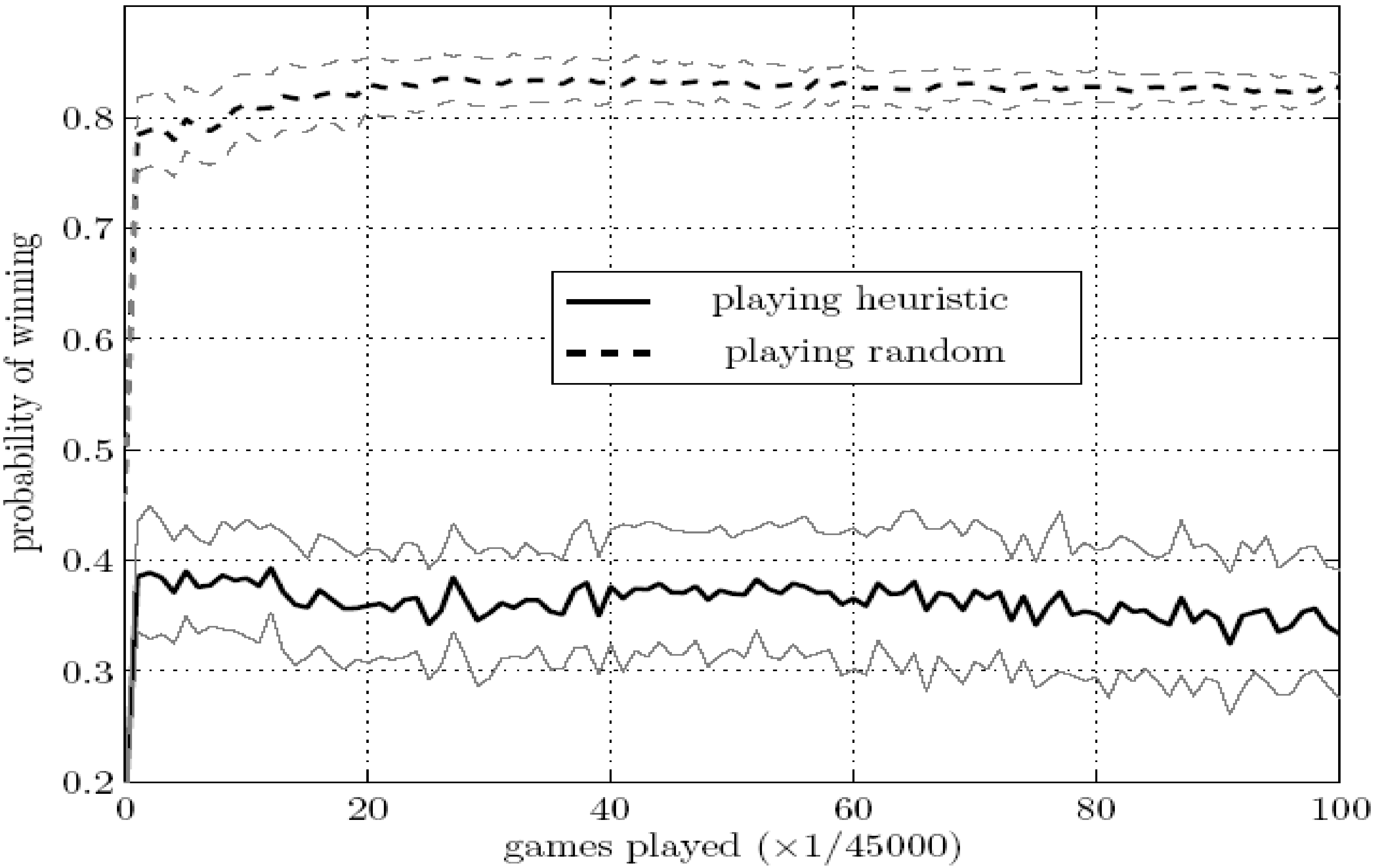
Algorithm in detail (Lucas and Runarsson, CIG 2006)

Initialize: $\boldsymbol{w}' = \mathbf{0}$ and $\beta = 0.05$ (or 1.0)
  while *termination criteria not satisfied* **do**
    for $k := 1$ to λ **do** (*replication*)
      $\boldsymbol{w}_k \leftarrow \boldsymbol{w}' + \boldsymbol{N}(0, 1/n)$
    od
    each individual $\boldsymbol{w}_k$, $k = 1, \ldots, \lambda$ *plays another*
     (once each color) for a total of $\lambda(\lambda - 1)$ *games,*
    find the player i *with the highest score (breaking ties randomly)*
    $\boldsymbol{w}' \leftarrow \boldsymbol{w}' + \beta(\boldsymbol{w}_i - \boldsymbol{w}')$ (arithmetic average)
  od

CEL (1,10) v. Heuristic

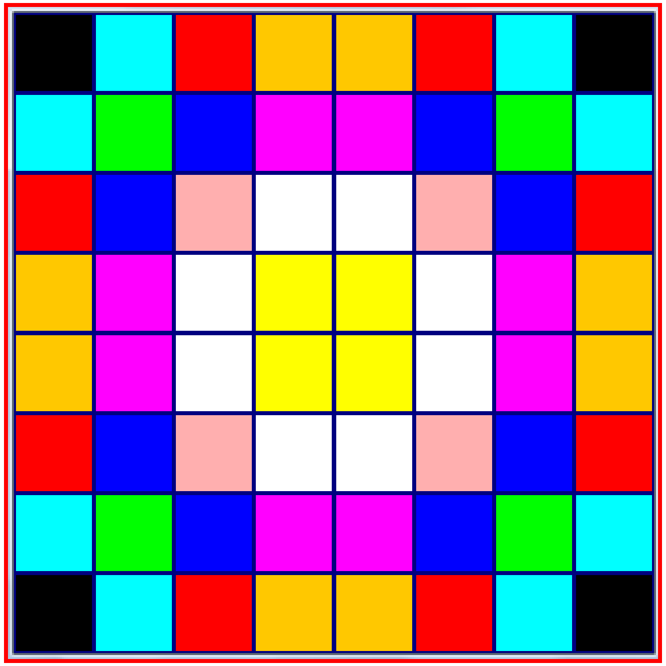


TDL v. Random and Heuristic



Othello: Symmetry

- Enforce symmetry
  - This speeds up learning



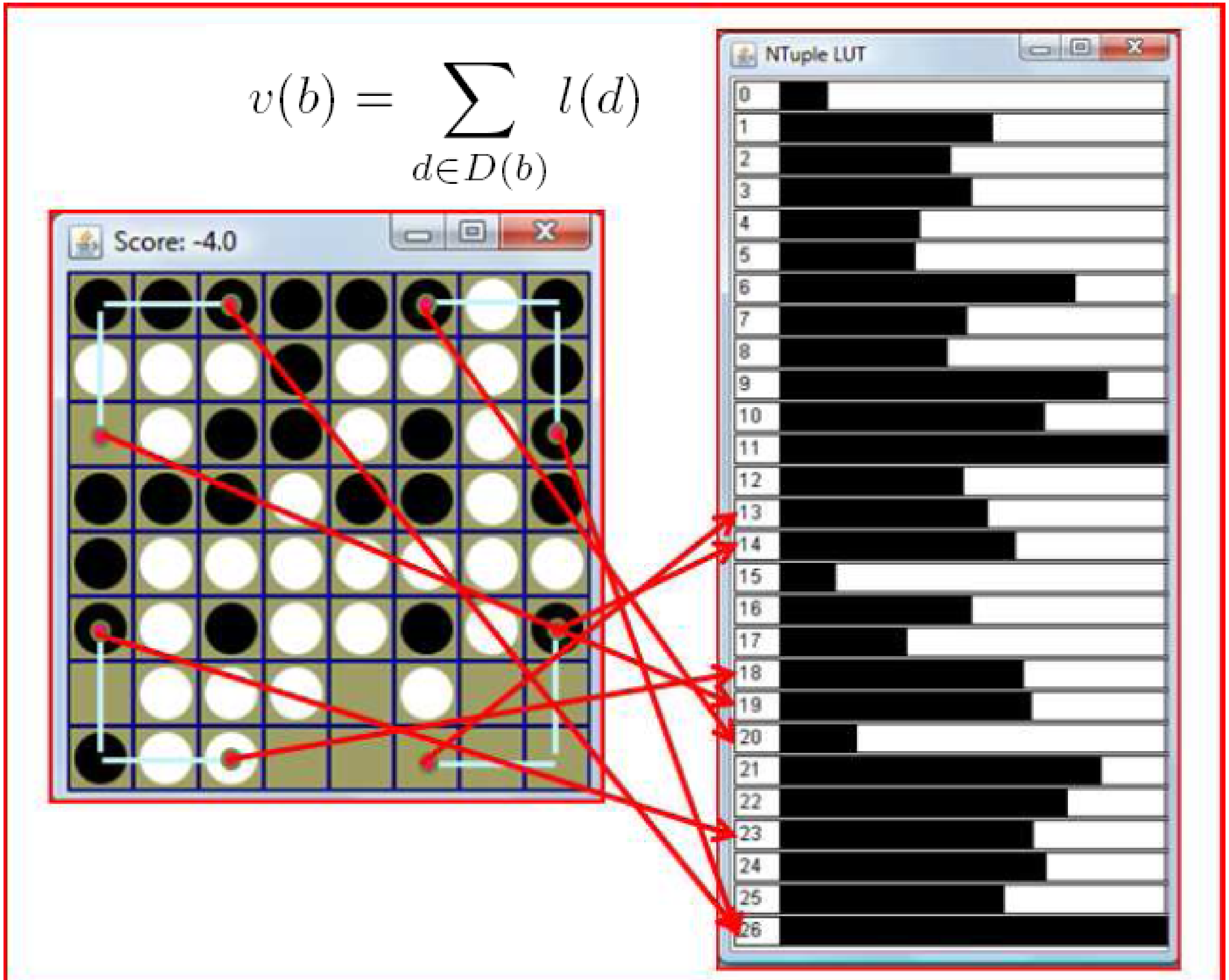
- Use trusty old friend: **N-Tuple System** for value approximator

NTuple Systems

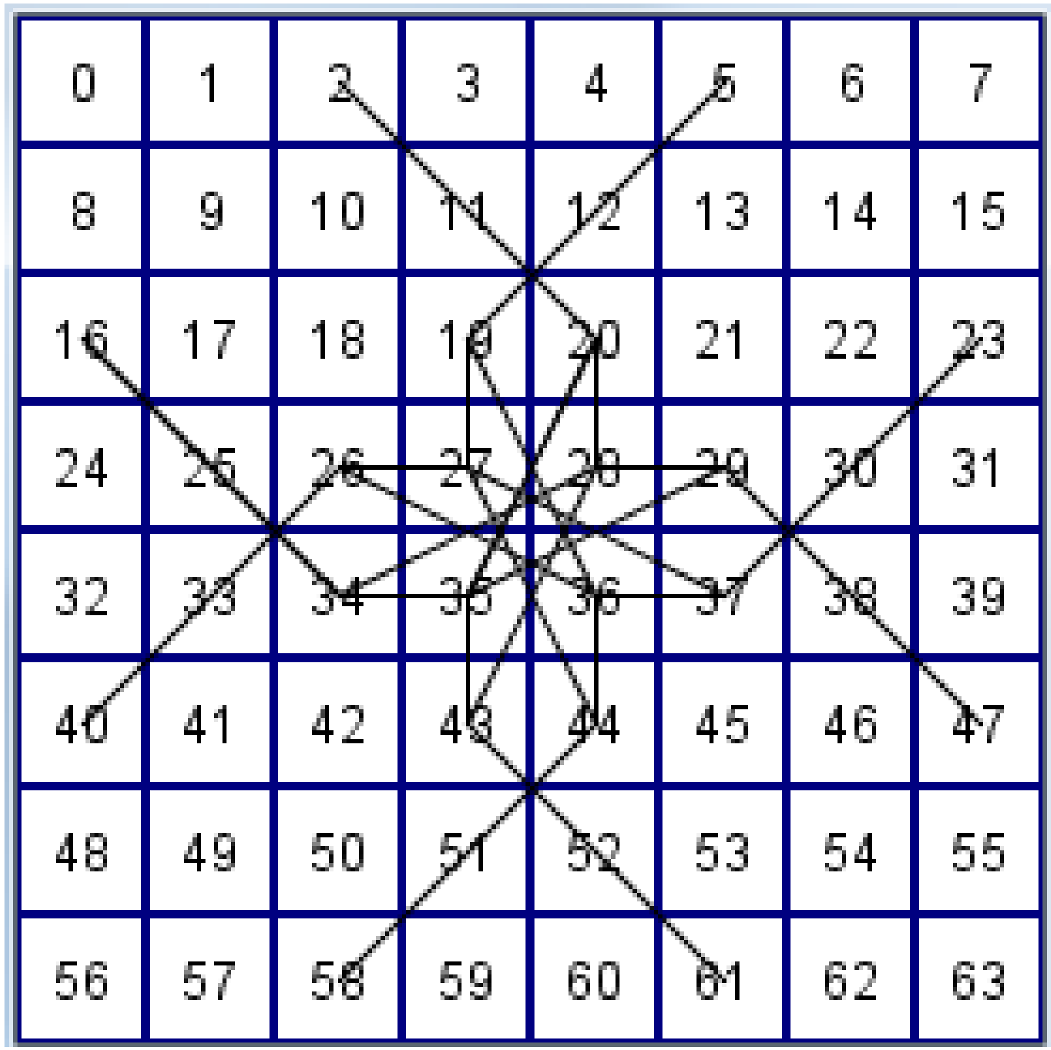
- W. Bledsoe and I. Browning. *Pattern recognition and reading by machine.* In Proceedings of the EJCC, pages 225 232, December 1959.
- Sample n-tuples of input space
- Map sampled values to memory indexes
  - Training: adjust values there
  - Recognition / play: sum over the values
- Superfast
- Related to:
  - Kernel trick of SVM (non-linear map to high dimensional space; then linear model)
  - Kanerva's sparse memory model
  - Also similar to Michael Buro's look-up table for Logistello

Symmetric 3-tuple Example

$$v(b) = \sum_{d \in D(b)} l(d)$$



Symmetric N-Tuple Sampling



N-Tuple System

- Results used 30 random n-tuples
- Snakes created by a random 6-step walk
  - Duplicates squares deleted
- System typically has around 15000 weights
- Simple training rule:

$$l(d) = l(d) + \delta \quad \forall\, d \in D(b)$$

N-Tuple Training Algorithm

Algorithm 2: N-tuple training algorithm

NOTE: f is the indexing function

INITIALIZE: set weights to zero

for *i in* set of n-tuples **do**

  for *j* in symmetries(i) **do**

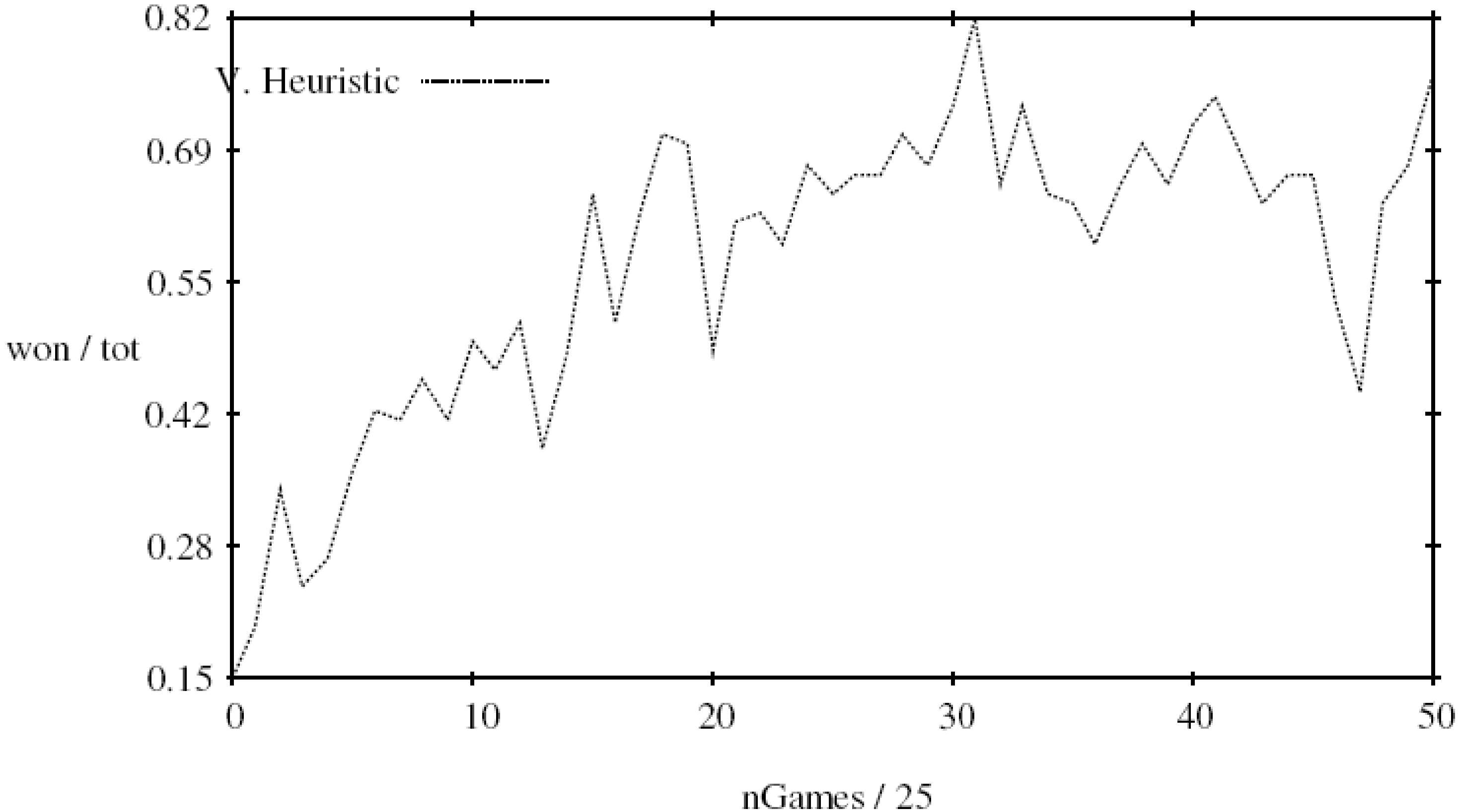
    index = f_{ij}(board)

    l_i[index] += δ

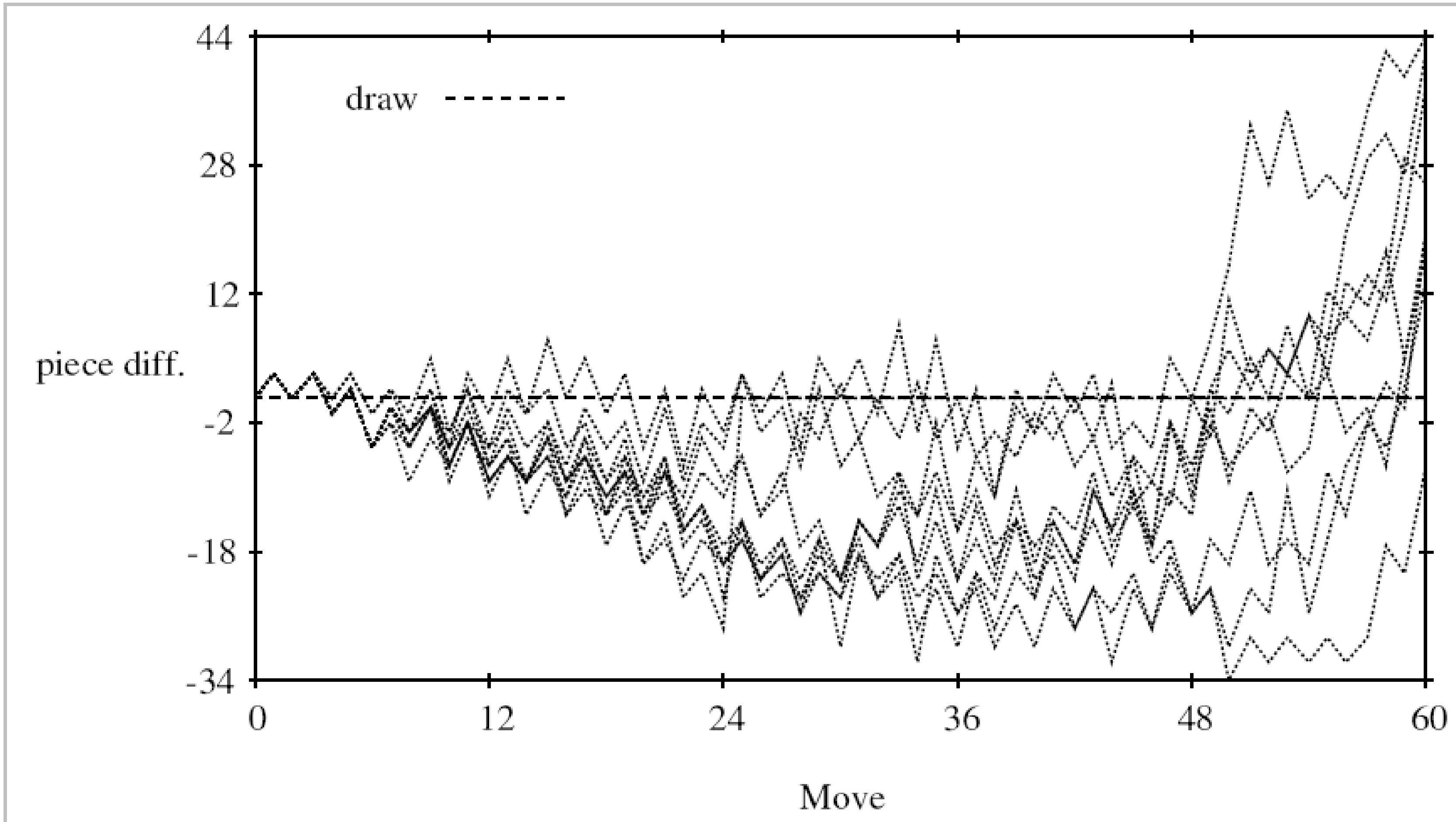
  end

end

NTuple System (TDL)
total games = 1250
(very competitive performance)



Typical Learned strategy… (N-Tuple player is +ve – 10 sample games shown)



Web-based League (May 15th 2008) All Leading entries are N-Tuple based

Trial League						
Position	**Name**	**Played**	**Won**	**Drawn**	**Lost**	**Format**
1	t15x6x8	100	79	3	18	SNT-Text
2	x30x6x8	100	71	4	25	SNT-Text
3	Stunner	100	67	1	32	SNT-Text
4	Woxy SNT	100	67	1	32	NET-WOX
5	WOX Test	100	65	1	34	NET-WOX
6	WOX Test 3	100	64	1	35	NET-WOX
7	newp8	100	64	3	33	SNT-Text
8	yp278a	100	64	2	34	SNT-Text
9	Stunner-2	100	63	6	31	SNT-Text
10	WOX Test 2	100	62	4	34	NET-WOX
11	MLP_Original-MoreNeurons.0.1-gen312-ties0.FF	100	60	4	36	MLP-Text
12	try3MLP_Original-MoreNeurons.0.1-gen341-ties0.FF	100	59	4	37	MLP-Text
13	shrd-MaxSolve-7c1kg	100	59	2	39	MLP-Text
14	test-mlp1	1000	582	34	384	unknown

Results versus CEC 2006 Champion (a manual EVO / TDL hybrid MLP)

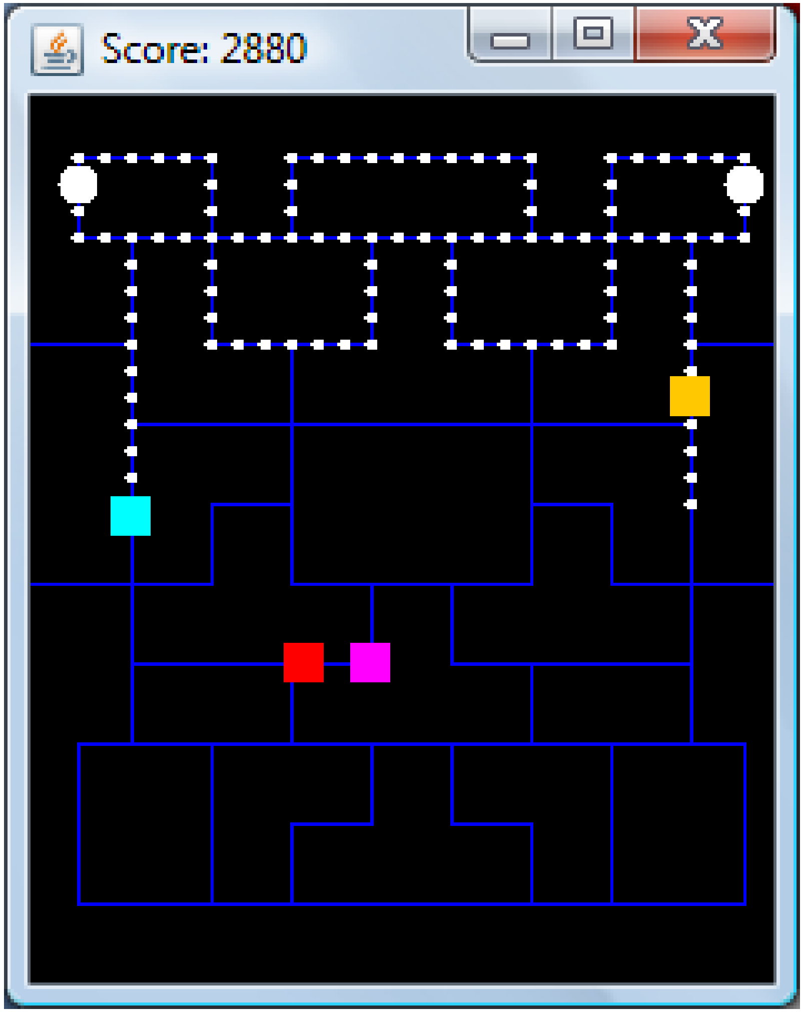
n_{sp}	Won	Drawn	Lost
250	89	5	106
500	135	6	59
750	142	5	53
1000	136	2	62
1250	142	5	53

N-Tuple Summary

- Stunning results compared to other game-learning architectures such as MLP
- How might this hold for other problems?
- How easy are N-Tuples to apply to other domains?

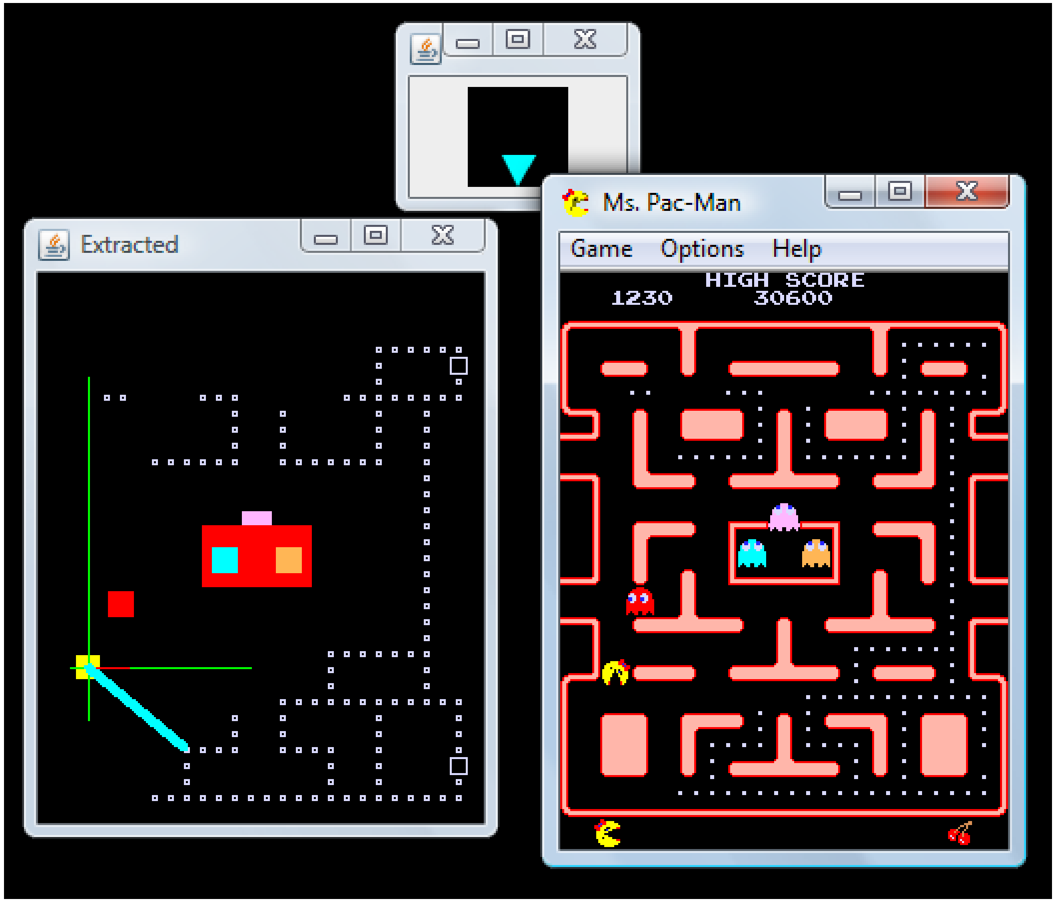
Ms Pac-Man

- Challenging Game
- Discrete but large search space
- Need to code inputs before applying to function approximator



Screen Capture Mode

- Allows us to run software agents original game
- But simulated copy (previous slide) is much faster, and good for training



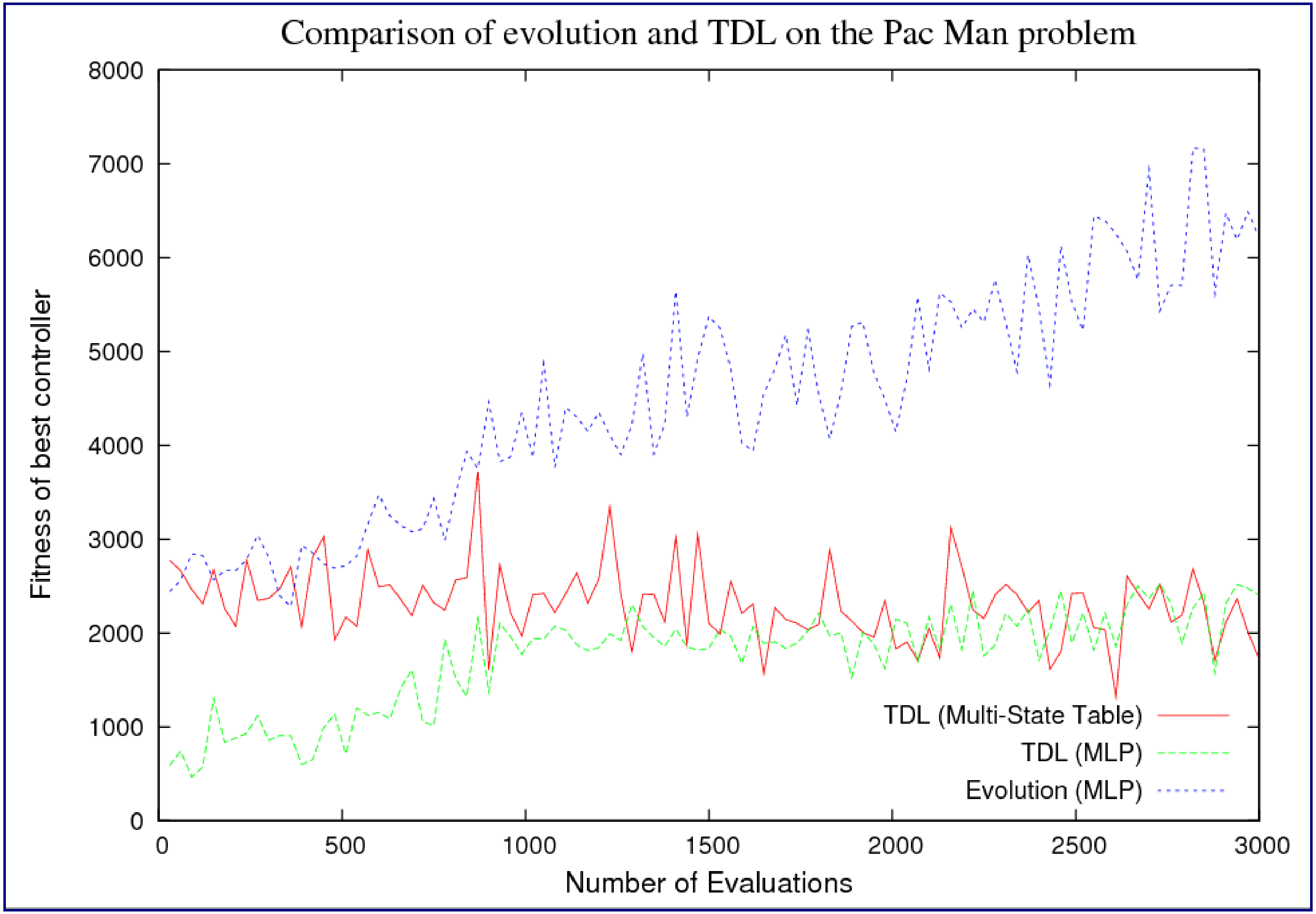
Ms Pac-Man Input Coding

- See groups of 4 features below
- These are displayed for each possible successor node from the current node
  - Distance to nearest ghost
  - Distance to nearest edible ghost
  - Distance to nearest food pill
  - Distance to nearest power pill

Alternative Pac-Man Features (Pete Burrow)

- Used a smaller feature space
- Distance to nearest safe junction
- Distance to nearest pill

So far: Evolved MLP by far the best!



Results: MLP versus Interpolated Table

- Both used a 1+9 ES, run for 50 generations
- 10 games per fitness evaluation
- 10 complete runs of each architecture
- MLP had 5 hidden units
- Interpolated table had 3^4 entries
- So far each had a mean best score of approx 3,700
- More work is needed to improve this
  - And to test transference to original game!

Summary

- All choices need careful investigation
  - Big impact on performance
- Function approximator
  - N-Tuples and interpolated tables: very promising
  - Table-based methods often learn much more reliably than MLPs (especially with TDL)
  - But: Evolved MLP better on Ms Pac-Man
    - Input features need more design effort…
- Learning algorithm
  - TDL is often better for large numbers of parameters
  - But TDL may perform poorly with MLPs
  - Evolution is easier to apply
- Some things work very well, though much more research needed
- This is good news!

New Transactions

IEEE TRANSACTIONS ON

COMPUTATIONAL INTELLIGENCE AND AI IN GAMES

A PUBLICATION OF THE COMPUTATIONAL INTELLIGENCE SOCIETY, THE IEEE COMPUTER SOCIETY, THE IEEE CONSUMER ELECTRONICS SOCIETY, AND THE IEEE SENSORS COUNCIL

www.ieee-cis.org/pubs/tciaig/

IEEE Computational Intelligence Society

IEEE computer society

CONSUMER ELECTRONICS SOCIETY

Sensors Council