Neural Networks camping

presented by VLM on 3. May 2005

Overview

- 1. Topologies
  - RBF
- 2. Learning methods
  - TDL
- 3. Application to Games
- 4. Considerations
- 5. Examples
  - Tic-Tac-Toe
  - Backgammon
  - Tigers and Goats (Asnyc. Game)
  - Chess
  - Go
  - CS

Topologies

Topologies

- Feed-Forward Net (Single/Multi)

Topologies

- Feed-Forward Net (Single/Multi)
- Recurrent Network

Topologies

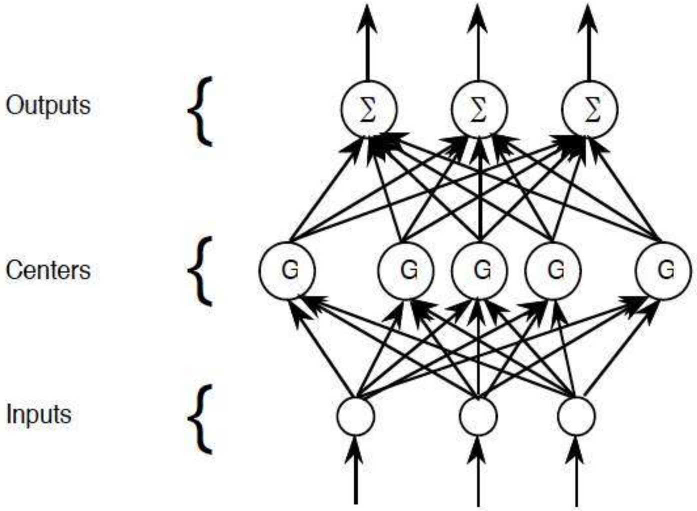
- Feed-Forward Net (Single/Multi)
- Recurrent Network
- SOM / SOTA

Topologies

- Feed-Forward Net (Single/Multi)
  - Radial Basis Function (RBF)
- Recurrent Network
- SOM / SOTA

Radial Basis Function

- 3 Layers
  - Inputs, RBF Units, Outputs
- Each RBF Unit has a center $\check{t}_i$ and a vector of coefficients $\check{c}_i$
- Value of Output unit is $y_j = \sum_{i=1}^{n} C_{ij} G(|\check{v} - \check{t}_i|^2)$
- G() is radial basis function, usually Gaussian $G(x) = e^{-\sigma x^2}$



- Number of RBF units equal to training examples with center set to input
- Reduces learning to only learning the coefficients
- Generalized RBF when allowed to have fewer centers

Learning

Learning

- Supervised ('learning with teacher' -> appxroximate I/O Mapping)
  -

Learning

- Supervised ('learning with teacher' -> appxroximate I/O Mapping)
  - Backpropagation (BP)
  - Backpropagation with Momentum
-

Learning

- **Supervised** ('learning with teacher'
  -> appxroximate I/O Mapping)
  - Backpropagation (BP)
  - Backpropagation with Momentum
- **Reinforcement learning** ('learning with critique'
  ->delayed reward / selfplay)
  - Genetic Algorithms (GA)
  - Temporal Difference Learning (TDL)
-

Learning

- **Supervised** ('learning with teacher' -> appxroximate I/O Mapping)
  - Backpropagation (BP)
  - Backpropagation with Momentum
- **Reinforcement learning** ('learning with critique' ->delayed reward / selfplay)
  - Genetic Algorithms (GA)
  - Temporal Difference Learning (TDL)
- **Unsupervised** (find underlying Properties)
  - Autoassociation – learns identity function
  - Time series prediction

TDL

- Compares each prediction to the following, and changes it
- Last prediction is compared to actual outcome
- Propagates error back from the last to the first
- Learns smoother prediction function
- Single-Step vs. Multi-Step Problem

TDL Algo

- Reminder: normal BP
- $\check{w} = \check{w} + \sum_{t=1}^{n} \Delta \check{w}_t$ (2.1)
- $\Delta \check{w}_t = \alpha (z - P_t) \nabla_w P_t$ (2.2)

- TDL uses normal FF-ANN
- Write Error as differences between succesive Predictions

  $(z - P_t) = \sum_{i=t}^{n} (P_{i+1} - P_i)$ with $P_{n+1} = z$

and replace using (2.1) & (2.2)

$$
\begin{aligned}
\check{w} = \check{w} + \sum_{t=1}^{n} \Delta \check{w}_t \quad &= \check{w} + \sum_{t=1}^{n} \alpha (z - P_t) \nabla_w P_t \\
&= \check{w} + \sum_{t=1}^{n} \alpha \sum_{k=t}^{n} (P_{k+1} - P_k) \nabla_w P_t \\
&= \check{w} + \sum_{k=1}^{n} \alpha \sum_{t=1}^{k} (P_{k+1} - P_k) \nabla_w P_t \\
&= \check{w} + \sum_{t=1}^{n} \alpha (P_{t+1} - P_t) \sum_{k=1}^{t} \nabla_w P_k
\end{aligned}
$$

- Which gives us $\Delta \check{w}_t = \alpha (P_{t+1} - P_t) \sum_{k=1}^{n} \nabla_w P_k$ (2.3)

TD(λ)

- (2.3) updates every prediction equally
- Preferable to affect more recent predictions more
- Introduce λ^k with $0 \leqslant \lambda \leqslant 1$:

$$\Delta \check{w}_t = \alpha (P_{t+1} - P_t) \sum_{k=1}^{t} \lambda^{t-k} \nabla_w P_k \qquad (2.4)$$

- (2.3) and (2.4) equal for λ = 1
- Thus (2.3) is TD(1)

- Corresponds to DFS: assumes most recent choices have most impact
- In Games BFS might also be reasonable: choices made early in the game determine outcome
  - Just invert λ: $\Delta \check{w}_t = \alpha (P_{t+1} - P_t) \sum_{k=1}^{t} \left(\frac{1}{\lambda}\right)^{t-k} \nabla_w P_k$

Application to Games

Application to Games

- Train Heuristic Function H(f)
-

Application to Games

- Train Heuristic Function H(f)
- Multiple Nets
  -

Application to Games

- Train Heuristic Function H(f)
- Multiple Nets
  - Train aspects (divide & conquer)
    - Train moves / actions
    - Train pieces
    - Train fields
  - Train judges and pick best
-

Application to Games

- Train Heuristic Function H(f)
- Multiple Nets
  - Train aspects (divide & conquer)
    - Train moves / actions
    - Train pieces
    - Train fields
  - Train judges and pick best
- Build NNTree as α-β Evaluator

Considerations

- One Net for both sides? (Async. Games)
- Training with Opponent?
  - Inferior – may not learn
  - Same level
  - Stronger – may learn to loose
- Circular states (repitition)
- Representation of input:
  - Wealthy (piece difference)
  - Plainly
  - Linear indepence of input vectors
    - NN might converge to shared point instead of any maximum

Example: TTT

- Learns H(f) by TD(0.6), α decreasing
- GRBF with 200 centers
  - 425 Iterations bootstrapping: 8 positions filled
  - 575 Iteratinons with 7 filled
  - Last 1000 1/5 of all positions as starting points
- 4 Experiments: selfplay, X, O and both against perfect opp.
- Input 10-dim Vector: 9 squares + turn
- Output 3-dim: P(X-win), P(O-win), P(draw)
- **Outcomes:** Play better when playing X
- Self-play best
- None learned underlying symmetry
- Only predicted draws accurately

Example: TDGammon

- Keys: Absolute vs. relative Error, stochastic
- MLP net, 198 inputs, 40-80 hidden, 3 outputs
- Self-play: every step calc. all dice rolls and play each resulting game
- After 300,000 games TDG 0.0 was as good as NeuroGammon

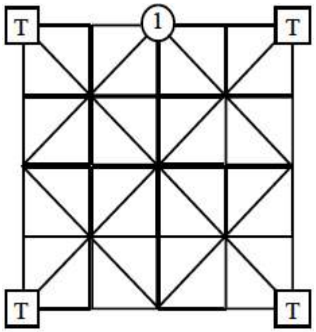
Programm	Hidden Units	Training Games	Opponents	Results
TDG 0.0	40	300000	Other Progs	Tied for best
TDG 1.0	80	300000	Robertie, Magriel...	-13/51 games
TDG 2.0	40	800000	Var. Grandmasters	-7/38 games
TDG 2.1	80	1500000	Robertie	-1/40 games
TDG 3.0	160	1500000	Kazaros	6/20 games

TDG 2.0 implemented 2-ply Tree-search

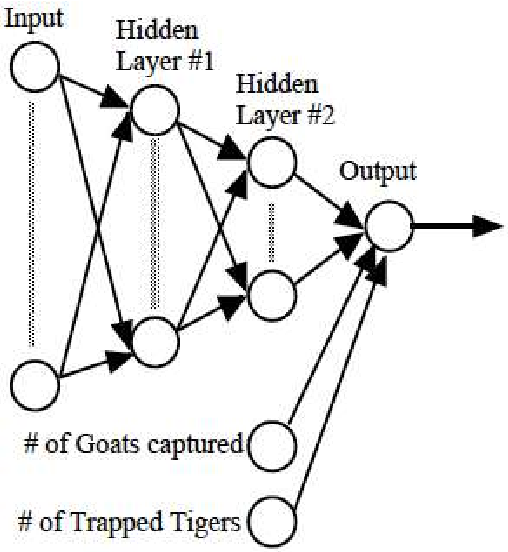
TDG 3.0 used selective 3-ply search

TDG 3.0 plays at grandmaster level and taught them how to play some posititions

Ex: Tigers&Goats



- FF-ANN (24,12,5,1), tanh transfer func:
  - Inputs include: #capt. Goats, #Tiger moves, #trapped Tigers, #goat moves w/o capt. Manh. Distance of each pair of Tigers
- Co-Evolution using GA
- 20 Networks for each side, each plays against 4 others, top 10 retained & mutated
- Results: Goats can at least draw the game
- Very complicated game for humans: admits no vague feelings about what features are correlated with good/bad position, maybe ANNs can help...



Ex: NeuroDraughts

- MLP with BPM
- Feature input better than plain board:
  - PieceAdvantage/DisAdv.
  - PieceThreat/Take
  - Our/his CenterControl
  - Mobility
  - Advancement

representation	wins	draws	losses	not lost	total
binary	133	258	189	391	580
direct	119	266	195	385	580
features	201	310	69	511	580

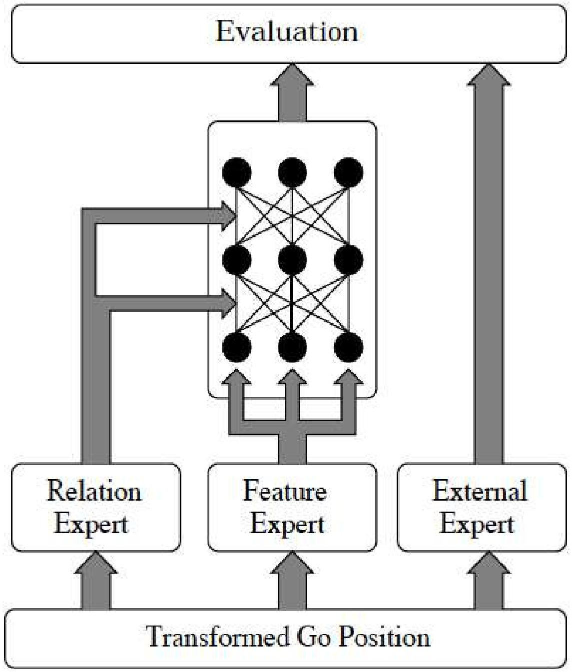
- **Findings:** Modular Net not advantageous
- Binary, direct Net very similar
- Direct I/O links stronger than FF-ANN
- Higher Discount value beneficial
- look-ahead affects λ, deeper search better than higher λ
- GA Co-Evolution with 2-ply search best, earlier clones perform sometimes better

Ex: NeuroChess

- Uses 2 ANNs
  - EBNN(175,165,175) trained first with a large grandmaster DB (120k games)
    - *Chess Model M* – captures domain spec. knowledge
    - Maps a board s_t to s_{t+2} 2 half-moves later
  - TD(0) then trains an evaluation network *V* (175,0-80,1)
    - 3-ply, Quiescence search
    - Uses M to bias its input
    - 90% Trained using grandmaster DB
    - Regularily played against GNUChess
- Weak opening
- Not as good as GNUChess and humans
- NN Evaluation takes longer than linear – less search time

Ex: NeuroGo

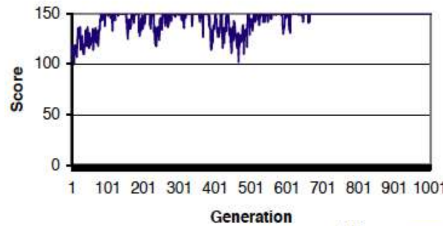
- FF-ANN, one unit per intersection
  - Bad for large boards
- Varying input hidden layer units (3-24)
- First board is transformed, connections are determined by Relation Expert (a-priori knowledge), mainly stone Dist.
- Ext. Expert operates solely, can override output of net, uses D. Benson's Algo

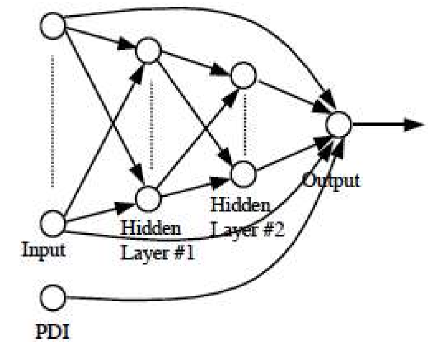


- Trained against itself, some P(move) as noise
- lost against 'Many faces of Go' which has a lot of feature knowledge

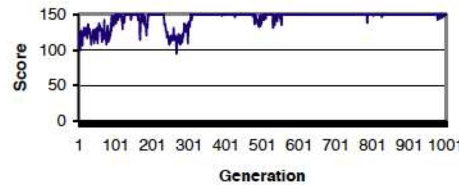
Ex: Kalah

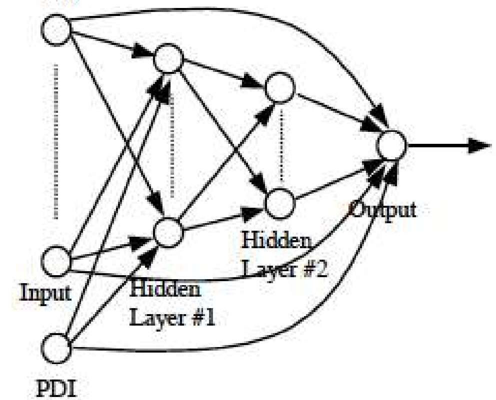
- Very similar to NeuroDraughts (Anaconda)
- FF-ANN(14,20,10,1)
- PDI is difference in Kalahs
- Co-Evolution, 1000 Gens., 5 opponents, top15
- **Results**:
  - (1) Direct-PDI



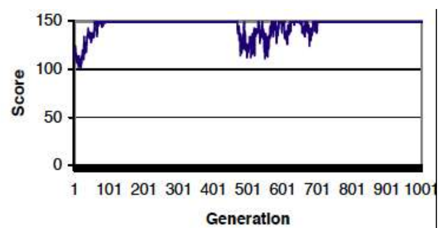


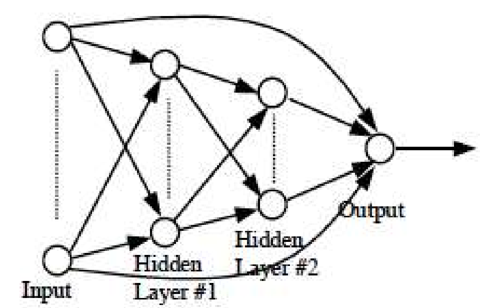
  - (2) Indirect-PDI





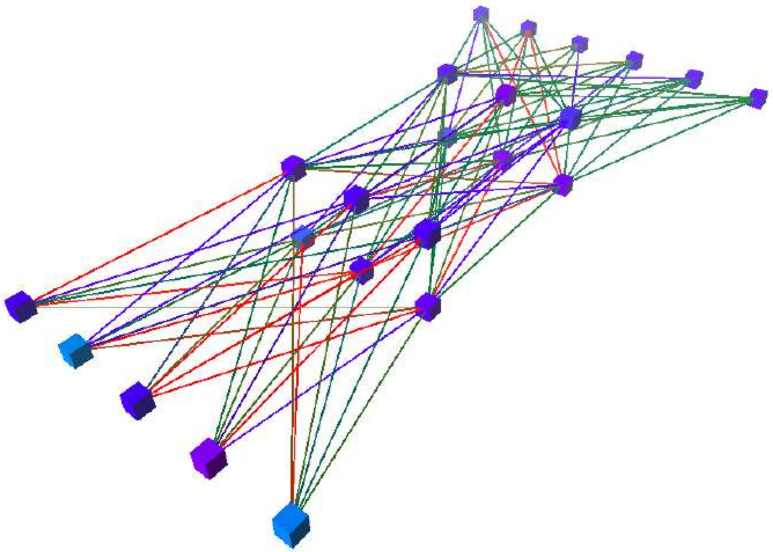
  - (3) No-PDI





- No-PDI not better than α-β, only wins as P1
- Indirect-PDI takes much longer to stabilize

Ex: CS - JoeBot



- Is trained offline, thus it can learn every strat.
- Online trained learned to 'camp'
- GA not doable, because Games to long

Kampf-NN is a FF-ANN(6,6,6,5), trained with BP

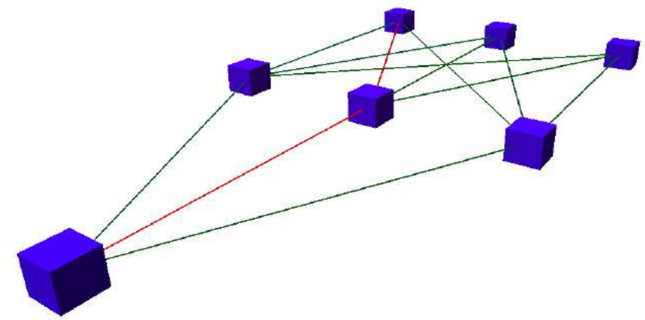
Inputs(-1..1): Health, Distance to Enemy, Enemy Weapon, Weapon, #Ammo, Situation (#Enemies,#team,mood)

Output: Jump, Duck, Hide, left/right, run/walk

3 memories:

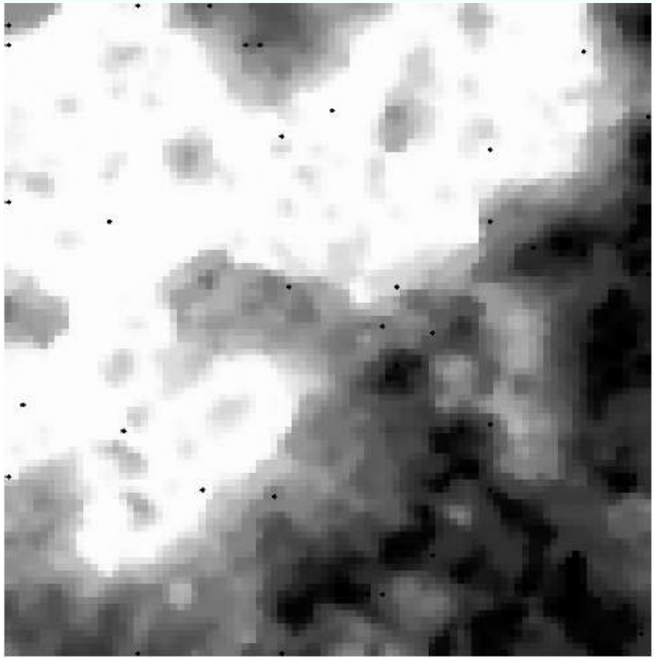
- Short-term: Enemies, 20secs
- Long-term: bomb, Enemies, gen. Things, RR(10)
- Waypoints and fights: sniping...

Ex: CS - JoeBot

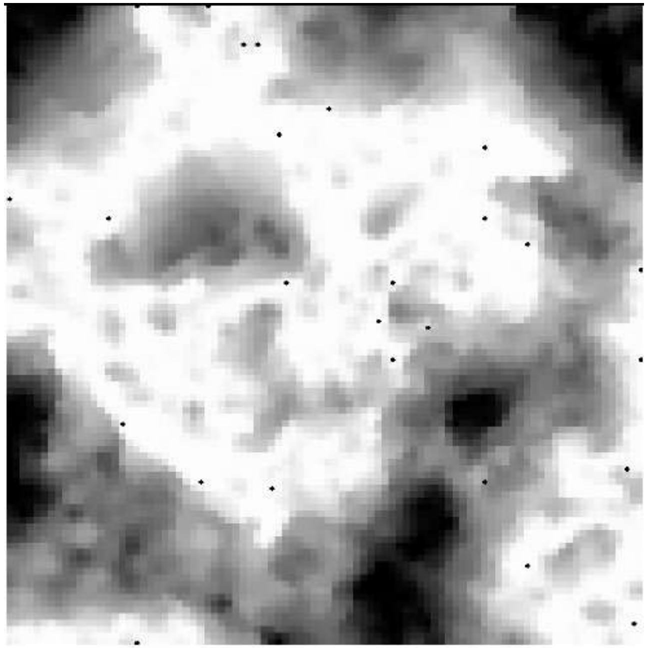


- Collision Net: FF-ANN(3,3,1)
- 3 Inputs are sensors in game (75 units long, 35°)
- Output: left/right

Ex: CS - JoeBot



Health SOM



Distance SOM

- Number of Training instances too large
- Instead capture all inputs to FFN during a game and give them to a SOM
- Look for differences that are very large, i.e. which the Net does not know too well, and manually retrain them
- Pics are from SOM(90,100), 12k training inst., stop at d<1 (438 epochs), P3-500Mhz ca. 31h