Learning Games using a Single Developmental Neuron

Gul Muhammad Khan[1] and Julian F. Miller[2]

[1]NWFP UET Peshawar, Pakistan, gk502@nwfpuet.edu.pk
[2]University of York, UK
jfm7@ohm.york.ac.uk

Abstract

An agent controlled by a single developmental neuron is trained to play arcade game. Genetic programming is used to find the DNA of neuron such that it can learn and store the learned information in the form of development in its architecture and updates in chemical concentration. The developmental neuron consists of dendrites, axons, and synapses that can grow, change and die. The structure of this neuron complexify itself at runtime as a result of game scenarios. The network is tested in arcade game environment of checkers. The agent has to recognize the patterns of the board and use this information to learn how to play the game better. The network is evolved against a professional checker program for its capability to learn. Input from the board is provided using sensory neuron through synapses. The developmental neuron process these signals and send output to the motor neurons to make a move. The structure of the neuron is also modified during signal processing. The developmental neuron successfully defeated the professional minimax based checker program during evolution by a large margin. We also tested the agent against some other opponents (not seen during evolution) of various levels for its generality and it proves to outperform them.

Introduction

In this paper we present the idea of developmental neuron capable of learning and adaptation. We have adopted the view that the intelligent behaviour of human being is the consequence of the special DNA. It is the DNA that is responsible for development of human body and brain. DNA of humans are different from other organisms that is why human can interact with each others. We beleive if we somehow manage to identify the functionality of human DNA and provide it with a neuron like structure we will be able to produce intelligent behaviour. Learning in brain is the consequence of biological development thus if we somehow manage to identify the rules for development we would be able to produce a learning system. DNA does not in itself encode learned information. Recent results demonstrated that even a single neuron has the capability of learning and adaptation as evident from the experimental results on snail aplysia (Kandel et al. (2000)). We have used Cartesian Genetic Programming (CGP) to develop a neuron having branching structure (Miller and Thomson (2000)). CGP represent the genotype (DNA) inside neuron responsible for development and signal processing. We evolved genotypes that encode programs that *when executed* gives rise to a neuron with developmental structure that can play checkers at higher level. The developmental and signalling functions are distributed at various segments (soma, axon, dendrite) inside neuron similar to biology (Zubler and Douglas (2009)).

We have produced an artificial agent that used this developmental neuron as its computational system. The agent receive information from checkers board using sensory neurons. Sensory neuron has a number of axonal branches that are distributed in the vicinity of CGP neuron and provide signal to them by making synapse. Synaptic transformation of signal is done using a CGP program similar to the one inside DNA of CGP neuron. CGP neuron recieves the external information in the form of its dendrite branches potential updates. This signal is then processed by CGP neuron using its DNA and a decision signal is transferred in forward direction to the motor neuron having dendrite branches distributed in the vicinity of CGP neuron.

The genotype inside CGP developmental neuron is a set of computational functions that are inspired by various aspects of biological neurons. Each agent (player) has a genotype that grows a computational neural structure (phenotype). The initial genotype that gives rise to the dynamic neural structure is obtained through evolution. As the number of evolutionary generations increases the genotypes develop structure that allow the players to play checkers increasingly well.

We have used an indirect encoding scheme in which the rules of network (CGP Neuron) are evolved instead the network directly. When we run these evolved programs they can adjust the network indefinitely. This allows our network to learn while it develops during its lifetime. The network begins as small randomly defined structure of neuron with dendrites and axosynapses. The job of evolution is to come up with genotypes that encode *programs* that when *executed* develop into mature neural structures that learn through environmental interaction and continued development. So the

complexity of the evolved programs is independant of the complexity of the task. The network continue to develop and complexify itself based on the environmental conditions.

A number of indirect methods are used in ANNs that evolve the rules for development of the network. ANN although inspired by biological nervous system has only few notions of biological brain. Here we have extended the view and identified a number of other important features that need to be added to individual neuron structure. These features prove to be extremly important for learning and memory. Memory and learning in brain is caused by many other mechanisms. Synaptic weights are only responsible for extremely short term memory (Kleim et al. (1998)), long term memory is stored in the structure of the neuron (Terje (2003)). The network presented here is an inspiration of biology, not the implementation of biology.

We have evolved the genetic programs inside CGP neuron that develop during the course of the game playing against a fixed level minimax program that plays checkers. At the start, the genes of neuron were random so the neuron behaviour was not that good during the course of game. As evolution progresses, the genes started to develop the neuron from an initial random structure such that it can understand the pattern of the board and use this information to make various intelligent moves such that it can beat a human intelligent based computer program. The opponent make moves based on the intelligence of humans who developed the program whereas the CGP developmental neuron evolved the inteligent genes that can cause a developmental neural structure that is capable of understanding the pattern of the board and play a move. The agent with a single neuron make a number of intelligent moves before it beat the opponent. These results prove that it is possible to evolve the genes that can produce networks capable of learning and intelligent decision making. To date, not a single developmental system proved to be capable of learning behaviour. This is the first time in the history of computational evolution that learning genes are evolved. The neuron structure continue to develop and change during the game. The results presented in paper clearly demonstrate that the learning capability of the agents improves over the course of evolution.

Cartesian Genetic Programming (CGP)

CGP is a well established and effective form of Genetic Programming. It represents programs by directed acyclic graphs (Miller and Thomson (2000)). The genotype is a fixed length list of integers, which encode the function of nodes and the connections of a directed graph. Nodes can take their inputs from either the output of any previous node or from a program input (terminal). The phenotype is obtained by following the connected nodes from the program outputs to the inputs. We have used function nodes that are variants of binary if-statements known as 2 to 1 multiplexers (Miller and Thomson (2000)). Multiplexers can be considered as atomic in nature as they can be used to represent any logic function (Miller and Thomson (2000)).

In CGP an evolutionary strategy of the form $1 + \lambda$, with λ set to 4 is often used (Miller and Thomson (2000)). The parent, or elite, is preserved unaltered, whilst the offspring are generated by mutation of the parent. If two or more chromosomes achieve the highest fitness then *newest* (genetically) is always chosen.

Developmental Models of Neural Networks

A number of developmental techniques are introduced to capture the learning capabilities by having time dependent morphologies. Nolfi et al presented a model in which the genotype-phenotype mapping (i.e. ontogeny) takes place during the individual's lifetime and is influenced both by the genotype and by the external environment (Nolfi et al. (1994)).

Cangelosi proposed a related neural development model, which starts with a single cell undergoing a process of cell division and migration until a neural network is developed (Cangelosi et al. (1994)). The rules for cell division and migration is specified in genotype, for a related approach see (Gruau (1994)).

Rust and Adams devised a developmental model coupled with a genetic algorithm to evolve *parameters* that grow into artificial neurons with biologically-realistic morphologies. They also investigated activity dependent mechanisms so that neural activity would influence growing morphologies (Rust et al. (1997)).

Federici presented an indirect encoding scheme for development of a neuro-controller (Federici (2005)). The adaptive rules used were based on the correlation between post-synaptic electric activity and the local concentration of synaptic activity and refractory chemicals.

Roggen et al. devised a hardware cellular model of developmental spiking ANNs (Roggen et al. (2007)). Each cell can hold one of two types of fixed input weight neurons, excitatory or inhibitory each with one of 5 fixed possible connection arrangements to neighbouring neurons. In addition each neuron has a fixed weight external connection. The neuron integrates the weighted input signals and when it exceeds a certain membrane threshold it fires. This is followed by a short refractory period. They have a leakage which decrements membrane potentials over time.

In almost all previous work the internal functions of neurons were either fixed or only parameters were evolved. Connections between neurons are simple wires instead of complicated synaptic process. The model we propose is inspired by the characteristics of real neurons.

Key features and biological basis for the model

Features of biological neural systems that we think are important to include in our model(Cartesian Genetic Programming Developmental Neuron (CGPDN)) are synaptic trans-

mission, and synaptic and developmental plasticity. Signalling between biological neurons happens largely through synaptic transmission, where an action potential in the pre-synaptic neuron triggers a short lasting response in the post-synaptic neuron (Shepherd (1990)). In our model signals received by a neuron through its dendrites are processed and a decision is taken whether to fire an action potential or not.

Neurons in biological systems are in constant state of change, their internal processes and morphology change all the time based on the environmental signals. The development process of the brain is strongly affected by external environmental signals. This phenomenon is called Developmental Plasticity. Developmental plasticity usually occurs in the form of synaptic pruning (Van Ooyen and Pelt (1994)). This process eliminates weaker synaptic contacts, but preserves and strengthens stronger connections. More common experiences, which generate similar sensory inputs, determine which connections to keep and which to prune. More frequently activated connections are preserved. Neuronal death occurs through the process of apoptosis, in which inactive neurons become damaged and die. This plasticity enables the brain to adapt to its environment.

A form of developmental plasticity is incorporated in our model, branches can be pruned, and new branches can be formed. This process is under the control of a 'life cycle' chromosome (described in detail in section 6) which determines whether new branches should be produced or branches need to be pruned. Every time a branch is active, a life cycle program is run to establish whether the branch should be removed or should continue to take part in processing, or whether a new daughter branch should be introduced into the network.

Starting from a randomly connected network, we allow branches to navigate (Move from one grid square to other, make new connections) in the environment, according to the evolutionary rules. An initial random connectivity pattern is used to avoid evolution spending extra time in finding connections in the early phase of neural development.

Changes in the dendrite branch weight are analogous to the amplifications of a signal along the dendrite branch, whereas changes in the axon branch (or axo-synaptic) weight are analogous to changes at the pre-synaptic level and post-synaptic level (at synapse). Inclusion of a soma weight is justified by the observation that a fixed stimulus generates different responses in different neurones.

Through the introduction of a 'life cycle' chromosome, we have also incorporated developmental plasticity in our model. The branches can self-prune and can produce new branches to evolve an optimized network that depends on the complexity of the problem (Van Ooyen and Pelt (1994)).

The CGP Neuron

This section describes in detail the structure and processing inside the CGP Neuron and the way inputs and outputs are interfaced with it.

The CGP Neuron is placed at a random location in a two dimensional spatial grid (as shown in figure 1). It is initially allocated a random number of dendrites, dendrite branches, one axon and a random number of axon branches. Neurons receive information through dendrite branches, and transfer information through axon branches to neighbouring dendrite branches. The branches may grow or shrink and move from one grid point to another. They can produce new branches and can disappear. Axon branches transfer information only to dendrite branches in their proximity. Electrical potential is used for internal processing of neurons and communication between neuron and is represented by an integer.

Health, Resistance, Weight and Statefactor

Four variables are incorporated into the CGP Neuron, representing either fundamental properties of the neuron (*health*, *resistance*, *weight*) or as an aid to computational efficiency (*statefactor*). The values of these variables are adjusted by the CGP programs. The *health* variable is used to govern replication and/or death of dendritic and axonal connections. The *resistance* variable controls growth and/or shrinkage of dendrites and axons. The *weight* is used in calculating the potentials in the network. Each soma has only two variables: *health* and *weight*. The *statefactor* is used as a parameter to reduce computational burden, by keeping neuron and branches inactive for a number of cycles. Only when the *statefactor* is zero are the neuron and branches are considered to be active and their corresponding program is run. *Statefactor* is affected indirectly by CGP programs.

Inputs, Outputs and Information Processing

The signal is transferred to and taken from this neuron using virtual axon and dendrite branches by making synaptic connections.

The signal from the environment is applied to CGP neuron using virtual input axo-synaptic connections. There are also virtual output dendrite branches used as the output of the system. The virtual axo-synaptic branches are allowed to not only transfer signals to the dendrite branches of processing neuron (CGP Neuron) but also to the output virtual dendrite branches which is the output of the system. The CGP Neuron transfers signals to the virtual output dendrite branches using the program encoded in the axo-synaptic chromosome.

Information processing in the CGP Neuron starts by selecting the list of dendrites and running the electrical dendrite branch program. The updated signals from dendrites are averaged and applied to the soma program along with the soma potential. The soma program is executed to get the final value of soma potential, which decides whether a neuron should fire an action potential or not. If soma fires, an action potential is transferred in forward direction using axo-synaptic branch programs.

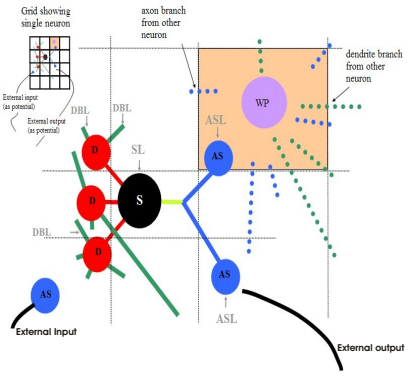

Figure 1: On the top left a grid is shown containing a single neuron. The rest of the figure is an exploded view of the neuron is given. Electrical processing parts containing dendrite (D), soma (S) and axo-synapse branch (AS) is shown as part of neuron. Developmental programs responsible for the 'life-cycle' of neural components (shown in grey) are also shown. They are dendrite branches (DBL), soma (SL) and axo-synaptic branches (ASL). The weight processing (WP) block shown is used to adjusts synaptic and dendritic weights.

Functionality of CGP Neuron

Neural functionality is divided into three major categories: electrical processing, life cycle and weight processing. These categories are described in detail below.

Electrical Processing

The electrical processing part is responsible for signal processing inside neuron and communication between neurons. It consists of dendrite branch, soma, and axo-synaptic branch electrical chromosomes.

The dendrite program D, handles the interaction of dendrite branches belonging to a dendrite. It take active dendrite branch potentials and soma potential as input and updates their values. The *Statefactor* is decreased if the update in potential is large and vice versa. If any of the branches are active (has its statefactor equal to zero), their life cycle program (DBL) is run, otherwise D continues processing the other dendrites.

The soma program S, determines the final value of soma potential after receiving signals from all the dendrites. The processed potential of the soma is then compared with the threshold potential of the soma, and a decision is made whether to fire an action potential or not. If it fires, it is kept inactive (refractory period) for a few cycles by changing its *statefactor*, the soma life cycle chromosome (SL) is run, and the firing potential is sent to the other neurons by running the AS programs in axon branches. AS updates neighbouring dendrite branch potentials and the axo-synaptic potential. The *statefactor* of the axosynaptic branch is also updated. If the axo-synaptic branch is active its life cycle program (ASL) is executed.

After this the weight processing program (WP) is run which updates the *Weights* of neighbouring (branches sharing same grid square) branches.

Life Cycle of Neuron

This part is responsible for replication, death, growth and migration of neurite branches. It consists of three life cycle chromosomes responsible for the neurites development. The two branch chromosomes update *Resistance* and *Health* of the branch. Change in *Resistance* of a neurite branch is used to decide whether it will grow, shrink, or stay at its current location. The updated value of neurite branch *Health* decides whether to produce offspring, to die, or remain as it was with an updated *Health* value. If the updated *Health* is above a certain threshold it is allowed to produce offspring and if below certain threshold, it is removed from the neurite. Producing offspring results in a new branch at the same grid square connected to the same neurite (axon or dendrite). The soma life cycle chromosome produces updated values of *Health* and *Weight* of the soma as output.

The Game of Checkers

Throughout the history of AI research, building computer programs that play games has been considered a worthwhile objective. Shannon developed the idea of using a game tree of a certain depth and advocated using a *board evaluation function* (Shannon (1950)) that allocates a numerical score according to how good a board position is for a player. The method for determining the best moves from these is called minimax (Dimand and Dimand (1996)). Samuel used this in his seminal paper on computer checkers (Samuel (1959)) in which he refined a board evaluation function. The current world champion at checkers is a computer program called Chinook (Schaeffer (1996)), which uses deep minimax search, a huge database of end game positions and a handcrafted board evaluation function based on human expertise.

More recently, board evaluations functions for various games including Checkers have been obtained through Artificial Neural Networks (ANNs) and often evolutionary techniques have been used to adjust the weights (Chellapilla and Fogel (2001)).

Although the history of research in computers playing games is full of highly effective methods (e.g. minimax, board evaluation function), it is highly arguable that human beings use such methods. Typically they consider relatively few potential board positions and evaluate the favourability of these boards in a highly intuitive and heuristic manner. They usually learn during a game, indeed this is how, generally, humans learn to be good at any game. So the question arises: How is this possible? In our work we are interested in how an *ability to learn* can arise and be encoded in a genotype that *when executed* gives rise to a neural structure that can play a game well.

Experimental Setup

The experiment is organized such that an agent is provided with CGPDN as its computational network. It is allowed to play against a minimax based checker program (MCP). The initial population of five agents, each starting with a small randomly generated initial network and randomly generated genotypes. The genotype corresponding to the agent with the highest fitness at the end of the game is selected as the parent for the new population. Four offspring formed by mutating the parent are created. Any learning behaviour that is acquired by an agent is obtained through the interaction and repeated running of program encoded by the seven chromosomes within the game scenario.

The MCP always plays the first move. The updated board is then applied to an agent's CGPDN. The potentials representing the state of the board are applied to CGPDN using the axo-synapse(AS) chromosome. The agent CGPDN is run which decide about its move. The game continues until it is stopped. It is stopped if all its or opponent players are taken, or if the agent or its opponent can not move anymore, or if the allotted number of moves allowed for the game have been taken.

Inputs and outputs of the System

Input is in the form of board values, which is an array of 32 elements, with each representing a playable board square. Each of the 32 inputs represents one of the following five different values depending on what is on the square of the board (represented by I). Zero means empty square. $I = M = 2^{32} - 1$ means a king, (3/4)M means a piece, (1/2)M an opposing piece and (1/4)M an opposing king.

The board inputs are applied in pairs to all the sixteen locations in the 4x4 CGPDN grid (i.e. two input axo-synapse branches in every grid square, one axo-synapse branch for each playable position) as the number of playable board positions are 32 as shown in figure 2. Figure 2 shows how the CGPDN is interfaced with the game board, input axo-synapse branches are allocated for each playable board position. These inputs run programs encoded in the axo-synapse electrical chromosome to provide input into CGPDN (i.e. the axo-synapse CGP updates the potential of neighbouring dendrite branches).

Input potentials of the two board positions and the neighbouring dendrite branches are applied to the axo-synapse chromosome. This chromosome produces the updated values of the dendrite branches in that particular CGPDN grid square. In each CGPDN grid square there are two branches for two board positions. The axo-synapse chromosome is run for each square one by one, starting from square one and finishing at sixteenth.

Output is in two forms, one of the outputs is used to select the piece to move, and second is used to decide where that piece should move. Each piece on the board has an output dendrite branch in the CGPDN grid. All pieces are assigned

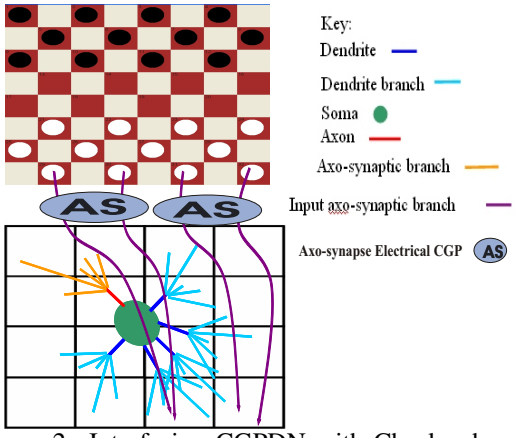

Figure 2: Interfacing CGPDN with Checker board. Four board positions are interfaced with the CGPDN such that board positions are applied in pair per square of CGPDN.

a unique ID, representing the CGPDN grid square where its branch is located. So the twelve pieces of each player are located at the first twelve grid squares. The player can only see its pieces, while processing a move and vice versa. Also the location of output dendrite branch does not change when a piece is moved, the ID of the piece represent the branch location not the piece location. Each of these branches has a potential, which is updated during CGPDN processing. The values of potentials determine the possibility of a piece to move, the piece that has the highest potential will be the one that is moved, however if any pieces are in a position to jump then the piece with the highest potential of those will move. In addition, there are also five output dendrite branches distributed at random locations in the CGPDN grid. The average value of these branch potentials determine the direction of movement for the piece. Whenever a piece is removed its dendrite branch is removed from the CGPDN grid.

CGP Developmental Neuron (CGPDN) Setup

The experiment parameters are arranged as follows. Each player CGPDN has a neuron with branches located in a 4x4 grid. Maximum number of dendrites is 5. Maximum number of dendrite are 200 and axon branches is 50. Maximum branch *statefactor* is 7. Maximum soma *statefactor* is 3. Mutation rate is 2%. Maximum number of nodes per chromosome is 200. Maximum number of moves is 20 for each player.

Fitness Calculation

The fitness of each agent is calculated at the end of the game using the following equation:

$$Fitness = A + 200(K_P - K_O) + 100(M_P - M_O) + N_M,$$

Where K_P represents the number of kings, and M_P represents number of men (normal pieces) of the player. K_O and M_O represent the number of kings and men of the opposing player. N_M represents the total number of moves

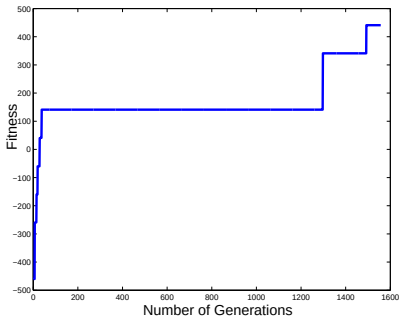

Figure 3: Fitness of CGPDN based player against MCP

played. A is 1000 for a win, and zero for a draw. To avoid spending much computational time assessing the abilities of poor game playing agents we have chosen a maximum number of moves. If this number of moves is reached before either of the agents win the game, then A =0, and the number of pieces and type of pieces decide the fitness value of the agent.

Results and Analysis

We have evolved agents against MCP in a number of evolutionary runs for 1500 generations and plotted it in figure 3. From the fitness graph, it is evident that the agent plays poorly at the early stage of evolution, but as the evolution progresses, the agent starts playing increasingly better and after 1250 generations, it begins to beat the opponent by three and four pieces margin. MCP is using minimax at ply level of 5. Agent plays with different strategy every time and finally manages to beat the opponent. It is worth mentioning here that the agent does not have any clue of what it is doing. It just receives signals from the board and produce moves accordingly, but as evolution progresses, the agent begins to understand the board and plays better. This is evident from the fitness graph shown in figure 3. Keeping in view that the agent is using a single neuron as a computational system and still manages to beat a program based on human (having trillion of neurons) intelligence is a big achievement demonstrated by any learning developmental system to date. Table 1 shows a game played between the well evolved agent and MCP. This is presented to demonstrate the level of play that the two players play. Figure 5 shows various stages of the game along with the corresponding neuron structure updated as a result of game scenario. Figure 4 shows the variation in the number of axon and dendrite branches of the CGP neuron during the game. Table 1 and figure 5 shows the complete game, the game start with black (MCP) making the first move by forwarding its piece from square 12 to 15. The updated board is applied as input to the CGPDN causing white(CGPDN) to forward a piece from square 21 to 17 as a result of signal received from CGPDN to motor neuron. Motor neuron receive signal using virtual dendrite branches distributed in the CGPDN Grid. Initially neuron has a small branching structure as evident from the first neuron image in figure 5 (Row-2, Column-1). Mutual exchange of pieces occur at various stages of the game and the neural branching structure continue to develop. The important break through occurs when black make a blunder at move-31 causing white to not only take two black pieces in one move but also becoming a king so that it can move both in forward and backward direction. Figure 5 show the move on the third row and last column. At this stage the CGPDN has the maximum dendrite branching structure so it can sense the signal from the board through its branches and act accordingly as evident from figure 5 and figure 4. The game continue until the aloted number of moves (40) are taken with white (CGPDN) having one king and a piece advantage over black(MCP).

Black Move	White Move
B1 12 - 15	W2 21 - 17
B3 10 - 13	W4 17 - 10
B5 5 - 14	W6 23 - 20
B7 1 - 5	W8 25 - 21
B9 14 - 19	W10 29 - 25
B11 5 - 10	W12 20 - 16
B13 10 - 13	W14 28 - 23
B15 19 - 28	W16 32 - 23
B17 13 - 17	W18 16 - 12
B19 7 - 16	W20 23 - 19
B21 15 - 20	W22 24 - 15
B23 11 - 20	W24 22 - 18
B25 8 - 12	W26 26 - 22
B27 17 - 26	W28 30 - 21
B29 9 - 13	W30 18 - 9
B31 2 - 5	W32 9 - 2 W33 2 - 11
B34 20 - 23	W35 27 - 20
B36 16 - 23	W37 22 - 18
B38 12 - 16	W39 11 - 14
B40 16 - 20	W41 19 - 15

Table 1: The first 41 moves of a game between a high evolved player (white) against MCP(black)

Generality

In order to test the generalization property of the agent, we have conducted a number of experiments by allowing the agent to play against five different opponents with various

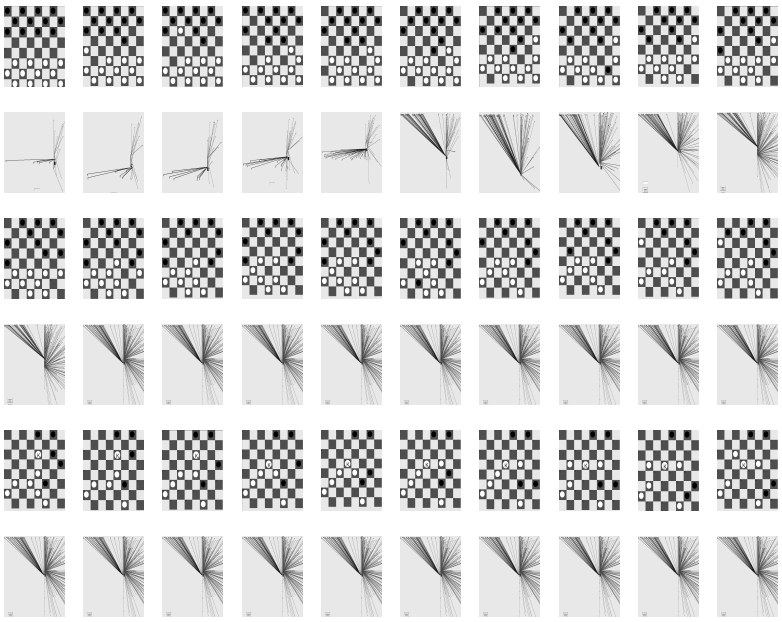

Figure 5: Various move played by CGP Neuron based agent and MCP with Agent playing white and MCP Black. Figure also shows the variation in neural structure during the game at various stages

Game Number	Winning Margin of CGPDN Agent	Level of opponent	Number of Moves to win
1	2 MEN and 1 King	50	76
2	2 MAN and 1 King	100	83
3	1 King	1000	111
4	1 MAN and 1 King	1200	120
5	lost by 1King and 4Men	1300	59

Table 2: Results of Evolved agent against various opponents not seen during evolution

playing levels. The neuron inside the agent starts with a random branching structure with the evolved genotype and continues to develop during the game. The agent was playing against completely new opponents that he has never played before during the course of evolution. Opponent's level of play is evident by the number of generations for which it is evolved. It beats the 50th generation agent by one King and two Men(normal peices) within 76 moves. An agent evolved for 100 generations also by one King and two Men but in 83 moves, the 1000th generation agent by one King in 111 moves and finally the 1200th generation by one Man and a King in 120 moves. In final case, the agent lost the game to a 1300 generations evolved player by one King and 4 Men in 59 Moves. It is worth mentioning that the agent was trained (evolved) to play forty moves. It never played a game beyond forty moves during evolution. From the re-

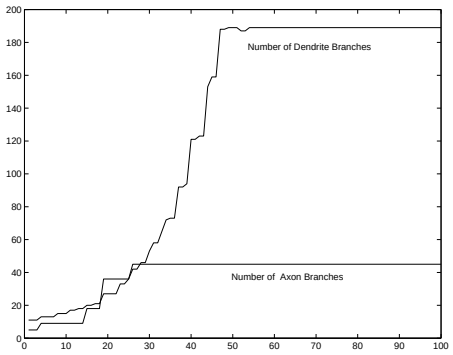

Figure 4: Variations in the number of Dendrite and Axon branches during the game

sults shown in table 2 it is evident that as the level of play of the opponent increases, the winning margin decreases, thus demonstrating clearly that we are able to obtain a DNA using CGP such that when used inside neuron produce a structure that can play game intelligently.

Conclusion

We have investigated the evolution of checkers playing agents that are controlled by a single developmental neuron. The development and signal processing inside neuron is controlled by a number of CGP programs working as DNA of the agent. The branching structure of neuron develops during the course of game. The agent demonstrated that it can play intelligently and beat a human intelligence based agent by a large margin. We have also tested a single neuron based agent for its generality. It beated the low level players with big margins in lesser time and tends to have problems beating high level players. From the results, it appears that we have successfully evolved CGP programs that encode *an ability* to learn 'how to play' checkers. In the future, we are planning to run the programs for longer, and against high level professional checkers agents to have more experience.

References

Cangelosi, A., Nolfi, S., and Parisi, D. (1994). Cell division and migration in a 'genotype' for neural networks. *Network-Computation in Neural Systems*, 5:497–515.

Chellapilla, K. and Fogel, D. B. (2001). Evolving an expert checkers playing program without using human expertise. In *IEEE Trans. on Evolutionary Computation*, volume 5, pages 422–428.

Dimand, R. W. and Dimand, M. A. (1996). A history of game theory: From the beginnings to 1945. *Routledge,Urbana*, 1.

Federici, D. (2005). Evolving developing spiking neural networks. In *Proceedings of CEC 2005 IEEE Congress on Evolutionary Computation*, pages 543–550.

Gruau, F. (1994). Automatic definition of modular neural networks. *Adaptive Behaviour*, 3:151–183.

Kandel, E. R., Schwartz, J. H., and Jessell, T. (2000). *Principles of Neural Science, 4rth Edition*. McGraw-Hill.

Kleim, J., Napper, R., Swain, R., Armstrong, K., Jones, T., and Greenough, W. (1998). Selective synaptic plasticity in the cerebellar cortex of the rat following complex motor learning. *Neurobiol. Learn. Mem.*, 69:274–289.

Miller, J. F. and Thomson, P. (2000). Cartesian genetic programming. In *Proc. of the 3rd European Conf. on Genetic Programming*, volume 1802, pages 121–132.

Nolfi, S., Miglino, O., and Parisi, D. (1994). Phenotypic plasticity in evolving neural networks. In *Proc. Int. Conf. from perception to action*. IEEE Press.

Roggen, D., Federici, D., and Floreano, D. (2007). Evolutionary morphogenesis for multi-cellular systems. *Journal of Genetic Programming and Evolvable Machines*, 8:61–96.

Rust, A. G., Adams, R., George, S., and Bolouri, H. (1997). Activity-based pruning in developmental artificial neural networks. In *Proc. of the European Conf. on Artificial Life (ECAL'97)*, pages 224–233. MIT Press.

Samuel, A. (1959). Some studies in machine learning using the game of checkers. *IBM J. Res. Dev.*, 3(3):210–219.

Schaeffer, J. (1996). *One Jump Ahead: Challenging Human Supremacy in Checkers*. Springer, Berlin.

Shannon, C. (1950). Programming a computer for playing chess. *Phil. Mag.*, 41:256–275.

Shepherd, G. (1990). *The synaptic organization of the brain*. Oxford Press.

Terje, L. (2003). The discovery of long-term potentiation. *Philos Trans R Soc Lond B Biol Sci*, 358(1432):617–20.

Van Ooyen, A. and Pelt, J. (1994). Activity-dependent outgrowth of neurons and overshoot phenomena in developing neural networks. *Journal of Theoretical Biology*, 167:27–43.

Zubler, F. and Douglas, R. (2009). A framework for modeling the growth and development of neurons and networks. *Frontiers in Computational Neuroscience*, 3.