In Search Of Intelligent Genes: The Cartesian Genetic Programming Computational Neuron (CGPCN)

Gul Muhammad Khan, Julian, F. Miller, David Halliday
NWFP UET Peshawar, Pakistan; *Intelligent System Design Group, University of York, UK.*
drgul@cgpneuron.com, jfm7@ohm.york.ac.uk, dh20@ohm.york.ac.uk.

Abstract— **Biological neurons are extremely complex cells whose morphology grows and changes in response to the external environment. Yet, artificial neural networks (ANNs) have represented neurons as simple computational devices. It has been evident for a long time that ANNs have learning abilities that are insignificant compared with some of the simplest biological brains. We argue that we understand enough neuroscience to create much more sophisticated models. In this paper, we report on our attempts to do this. We identify and evolve seven programs that together represents a neuron which grows *post evolution* into a complete 'neurological' system. The network that occurs by running the programs has a highly dynamic morphology in which neurons grow, and die, and neurite branches together with synaptic connections form and change. We have evaluated the capability of these networks for playing the game of checkers. Our method has no board evaluation function, no explicit learning rules and no human expertise at playing checkers is used. The learning abilities of these networks are encoded at a *genetic* level rather than at the phenotype level of neural connections.**

I. Introduction

The idea of constructing programs modeled on the brain is motivated by systems which exhibit intelligence, the capability for learning, and self adaptation. The brain has many highly desirable features that are hard to replicate in conventional computer systems, it is developmental, in that it acquires increasingly sophisticated capabilities over time and although constantly changing, yet it always retains its integrity as a learning system. It is adaptive, and shows plasticity to changes in its environment, so that new experiences and stimuli are incorporated into the neural system without altering existing capabilities. It shows tolerance to damage and the ability to self-repair and self-reorganize in such a way that it retains functionality, the brain is highly versatile in its ability to learn diverse tasks and to develop abstract symbolic models which enable the living system to operate effectively in complex environments.

It is apparent that although highly complex, the brain is made of essentially similar building blocks and that these blocks are themselves composed of highly interconnected networks of similar neurons. In the same way that the developmental engine of the biological cell contains the mechanism for building the human body, we believe that the key to the sophistication of the brain lies in the *developmental* power of the neuron.

The motivation behind the research of this work is to develop a system capable of learning and adapting itself to the environment. Artificial Neural Networks (ANNs), though inspired by the brain have largely ignored many aspects of biological neural systems [1]. Originally, there were good reasons for this. Simple models were required that could be executed on relatively slow computers. Also they were amenable to formal analysis. However, the computational power of modern computers has made more complex neuro-inspired approaches much more feasible. At the same time, our understanding of neuroscience has increased considerably. Important neglected aspects include neural development, neuron structure and mechanisms of communication between neurons. ANNs consider the brain as a connectionist system just like nodal network, and each neuron is considered as a node containing signal processing functions. Real neurons do complex processing through its neurite branches before the signal reaches the soma, where a decision about signal transformation is made based on the signals received through dendrites. Biological neurons are located in space and transfer signals to their neighbours through electrochemical signal by making synapse. These synaptic connections are not fixed and change over the course of time, also the branching structure of neurons and the number of neurons in the brain changes. These dynamic capabilities of the brain are responsible for its generalized capabilities of learning and adaptation. Our research goal is to produce a system which has capability of learning to learn. In this work we have used a particular form of GP called Cartesian Genetic Programming (CGP) [2]. Each neuron is considered as a computational device with each sub-processing part represented by a chromosome. The genotype of a neuron consists of a collection of chromosomes representing the sub-components of the neuron.

We have provided each neuron with a structural morphology such that it consists of a soma, dendrites [3], axons with branches and dynamic synapses [4] and synaptic communication. Neurons are placed in a two dimensional toroidal grid to give branches a sense of virtual proximity. Branches are allowed to grow and shrink, and communication between axon branches and dendritic branches is allowed.

To achieve this we have idealized seven essential neural

978-1-4244-2959-2/09/$25.00 © 2009 IEEE

components which we have represented as CGP chromosomes encoding combinational digital circuits [5]. These chromosomes encode distinct computational functions representing aspects of real neurons. This model allows neurons, dendrites, and axon branches to grow, die and change while solving a computational problem (when there is no evolution). Also the synaptic morphology can change and affect the information processing. While this model in undeniably quite complex and involves many variables and parameters we feel this is justified by the evident enormous complexity of the brain. The computational network that forms when the seven chromosomes are run (not evolved) grows a network of neurons, neurites and synapses that reflect its own internal dynamics and environmental interaction.

In our view the process of biological development underpins learning. Development is the time dependent process in which a system grows and is shaped by environmental interaction. It is evident that in biology this emergent process begins at a genetic level. This raises the question: how is a capability for learning encoded at a genetic level? In this work we have tried to answer this question on a classic problem in artificial intelligence, the game of Checkers.

Conventional ANNs represent neurons as static networks of extremely simple computational units that do not change their topology while being trained. ANNs were proposed when computers were in their infancy and were very slow, thus simple models were at the time, the only feasible models. However such static models are in marked contrast to what happens in real brains. Brains grow and acquire extremely sophisticated abilities over time. Indeed the most rapid learning occurs when brains are growing and changing (e.g. the learning of language). There is also abundant evidence from neuroscience that learning and memory are intimately related to time-dependent neural processes [6] some of which cause morphological changes in neurons [7]. In conventional ANNs, memory is encoded in the form of static weights but we know now that the location and mechanisms responsible for remembered information is not fixed and purely synaptic, but involves many mechanisms in constant (though, largely gradual) change [8]. Even dendrites themselves should no longer be regarded as passive entities that simply collect and pass synaptic inputs to the soma, and Koch argues that ”dendritic trees enhance computational power” [9]. Neurons communicate through synapses but synapses are not just a static strength of connection, they change in a signal dependent way over various time scales [7].

In this paper we are examining whether developmental programs can be evolved that create a neural network that can learn how to play checkers.

In 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*[10] that allocates a numerical score according to how good a board position is for a player. The method for determining best moves from these is called minimax [11]. Samuel used this in his seminal paper on computer checkers [12] in which he refined a board evaluation function. After two computer players have played a game, the loser is replaced with a deterministic variant of the winner by altering the weights on the features that were used, or by replacing features that had very low weight with other features. The current world champion at checkers is a computer program called Chinook[13]. 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 have been obtained through Artificial Neural Networks (ANNs) and often evolutionary techniques have been used to adjust the weights: Othello [14], Go [15], Chess [16], and Checkers [17].

We have three major criticisms of these approaches. The first criticism is in the use of a board evaluation function and the minimax algorithm. Such methods appears to bear little resemblance to the methods that human beings use to play games well. Typically, human beings 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 since we are interested in where learning comes from, we have rejected these approaches. Instead our networks will not return a numerical value for the favourability of a board position or use minimax but merely indicate which piece to move and where.

Our second criticism is with those methods that evolve weights to achieve a high standard of play. This has no biological plausibility. Firstly, natural evolution produces organisms that are gradually better adapted to their environment and this is an *extremely* slow process. Secondly, trying to evolve weights will inevitably become infeasible when the number of neural connections become very large. Thirdly, learning in organisms happens *in their lifetime* and evolution is *not* involved. We are interested in where this learning comes from, and so we are trying to use evolution to create the rules that construct a learning system. In this way the size of the genotype will be unrelated to the size and connectivity of the network. We emphasize that in our approach *no evolution* takes place while the programs play checkers.

Since we do not know how to design a computational neuron that constructs a learning system we use a method of automatic program evolution called Genetic Programming to discover this [18]. Our genotype is a set of seven chromosomes that encode programs that represent various aspects of real neurons [5]. As we will see, when the programs encoded in an agent's (player) genotype are executed they cause a computational neural structure to grow that can play checkers. The key idea here is that we do not evolve a neural network but we *evolve the programs that when executed* build and continuously shape and change the network at run time (when no evolution takes place).

Section 2 reviews previous work on the evolution of developmental processes that build ANNs. Section 3 describes the method we use to evolve the genotype in our system. Section 4 provides the key features and biological basis of the model. Section 5 describes the structure and operation of our computational network. We present our results on checkers in section 6 and then conclude the paper in section 7.

II. Developmental Artificial Neural Networks

One of the earliest attempts to use development in ANNs, evolved network architecture and the connection strengths for robot control [19]. They used a 2-dimensional space for the network, where a collection of artificial neurons are distributed with growing and branching axons. The genetic code specifies the instructions for axonal growth and branching. Connections between neurons are made when one axon reaches another neuron.

Cangelosi proposed a neural development model, which starts with a single cell undergoing a process of cell division and migration [20]. Each cell produces two daughter cells and division and migration continues until a collection of neurons is obtained. Neurons then grow axons to produce connections. The rules for cell division and migration are stored in the genotype. Gruau proposed an efficient developmental method for developing traditional neural networks [21] using a tree-based GP system. Rust et al. used a developmental model coupled with a genetic algorithm to evolve parameters that grow artificial neurons with biologically-realistic morphologies [22]. They also investigated activity dependent mechanisms [23] so that neural activity would influence growing morphologies. Although they showed that the technique was able to produce realistic and activity dependent morphologies of neurons, they did not investigate the networks carrying out a function.

Jakobi created an artificial genomic regulatory network [24] that used 'proteins' to define neurons in a recurrent ANN with excitatory or inhibitory dendrites for robot control. Federici presented an indirect encoding scheme for development of neuro-controllers, and compared it with a direct scheme [25]. The adaptive rules used were based on the correlation between post-synaptic electric activity and the local concentration of synaptic activity and simulated refractory chemicals. Astor and Adami's ANNs involved artificial chemistry [26]. In an attempt to avoid the computational expense of running programs representing dendritic and axonal branches, Downing favours a higher abstraction level which maintains key aspects of cell signalling, competition and cooperation of neural topologies in nature [27].

Although research into the use of development in ANNs has been valuable and interesting, it has, so far, been of a rather exploratory nature, with various authors building and evaluating models on small problems. To our knowledge there have been no developmental ANNs that have been applied to substantial problems in AI and Machine Learning. This was also one of the motivations for us to tackle checkers.

III. Cartesian Genetic Programming (CGP)

CGP is a well established and effective form of Genetic Programming. It represents programs by directed acyclic graphs [2]. 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. For our checkers work we have used function nodes that are variants of binary if-statements known as 2 to 1 multiplexers [28] as shown in figure 1.

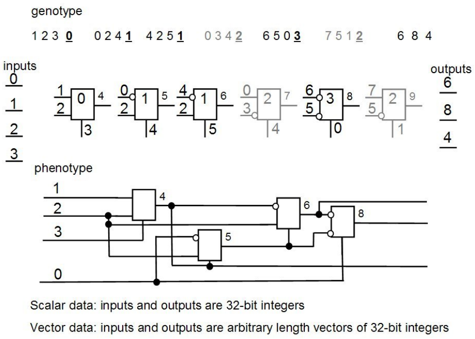

genotype
1 2 3 0 0 2 4 1 4 2 5 1 0 3 4 2 6 5 0 3 7 5 1 2 6 8 4
inputs
0
1
2
3
outputs
6
8
4
phenotype
Scalar data: inputs and outputs are 32-bit integers
Vector data: inputs and outputs are arbitrary length vectors of 32-bit integers

Fig. 1. Structure of CGP chromosome. Showing a genotype for a 4 input, 3 output function and its decoded phenotype. Inputs and outputs can be either simple integers or an array of integers. Note nodes and genes in grey are unused and small open circles on inputs indicate inversion. The function gene in genotype is underlined. All the inputs and outputs of multiplexers are labeled. Labels on the inputs of the multiplexer shows where are they connected (i.e. they are addresses). Input to CGP is applied through the input lines as shown in figure. The number of inputs (four in this case) and outputs (three in this case) to the CGP is defined by the user, which is different from the number of inputs per node (three in this case i.e. a, b and c.)

The four functions in figure 1 are the possible input combinations of a three input (two inputs and a control) multiplexer (see numbers in rectangles). Multiplexers can be considered as atomic in nature as they can be used to represent any logic function [28].

Figure 1 shows the genotype and the corresponding phenotype obtained connecting the nodes as specified in the genotype. The Figure also shows the inputs and outputs to the CGP. Output is taken from the nodes as specified in the genotype (6, 8, 4). In our case we have not specified the output in the genotype and have used a fixed pseudo random list of numbers to specify where the output should be taken from.

The evolutionary strategy utilized is of the form 1 + λ, with λ set to 4. The parent, or elite, is preserved unaltered, whilst the λ offspring are generated by mutation of the parent. The best chromosome is always promoted to the next generation, if two or more chromosomes achieve the highest fitness then *newest* (genetically) is always chosen [28].

IV. Key features and biological basis for the model

Features of biological neural systems that we think are important to include in our model are synaptic transmission, 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 [29]. 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. Table I lists all the properties of biological systems that are incorporated into our model. Table I also shows the presence and absence of these properties in existing ANNs and neural development models.

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 [30]. 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 [30].

V. The CGP Computational Network (CGPCN)

This section describes in detail the structure of the CGPCN, along with the rules and evolutionary strategy used to run the system[1].

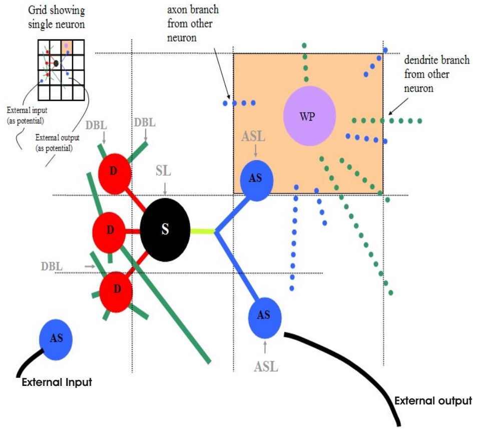

Grid showing single neuron
axon branch from other neuron
dendrite branch from other neuron
External input (as potential)
External output (as potential)
DBL
DBL
DBL
D
D
D
SL
S
ASL
AS
WP
AS
ASL
AS
External Input
External output

Fig. 2. On the top left a grid is shown containing a single neuron. The rest of the figure is an exploded view of the neuron. The neuron consists of seven evolved computational functions D, S, AS, DBL, SL, ASL, WP. D, S, and AS are electrical and process a simulated potential in the dendrite, soma and axo-synapse branch respectively. Three more (DBL, Sl and ASL) are developmental in nature and are responsible for the life-cycle of neural components (shown in grey). These respectively decide whether dendrite branches, soma and axo-synaptic branches should die, change, or replicate. The remaining evolved computational function (WP) adjusts synaptic and dendritic weights and is used to decide the transfer of potential from a firing neuron (dashed line emanating from soma) to a neighbouring neuron

In the CGPCN neurons are placed randomly in a two dimensional spatial grid so that they are only aware of their spatial neighbours (as shown in figure 2). Each neuron 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 neurons. The dynamics of the network also change since branches may grow or shrink and move from one CGPCN grid point to another. They can produce new branches and can disappear, and neurons may die or produce new neurons. Axon branches transfer information only to dendrite branches in their proximity. Electrical potential is used for internal processing of neurons and communication between neurons and we represent it as an integer.

Health, Resistance, Weight and Statefactor

Four variables are incorporated into the CGPCN, representing either fundamental properties of the neurons (*health*, *resistance*, *weight*) or as an aid to computational efficiency (*statefactor*). Biological neurons clearly have a health since neurons can become feeble and die. Neurites have electrical resistance which is related to the length. The weight is an analogue of their efficacy in transmitting signals. In our model, the values of these variables are adjusted by the CGP programs. The *health* variable is used to govern replication and/or death of dendrites and 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 some of the neurons and branches inactive for a number of cycles. Only when the *statefactor* is zero the

[1] For a complete description see the first authors PhD thesis, which is available at http://miller.jules.googlepages.com/PhD-thesis-GM-Khan.pdf

Name	ANNs	Neural development	Biology	CGPCN
Neuron Structure	Node with connections	Node with axons and dendrites	Soma with dendrites, axon and dendrite branches	Soma with dendrites, axon and dendrite branches
Interaction of branches	No	No	Yes	Yes
Neural function	Yes	Yes	Yes	Yes
Resistance	No	Yes/No	Yes	Yes
Health	No	No	Yes	Yes
Neural Activity	No	No	Yes	Yes
Synaptic Communication	No	No	Yes	Yes
Arrangement of Neurons	Fixed	Fixed	Arranged in space (Dynamic Morphology)	Arranged in Artificial space (Dynamic Morphology)
Spiking (Information processing)	Yes, but not all	Yes, but not all	Yes	Yes
Synaptic Plasticity	Yes	No	Yes	Yes
Developmental Plasticity	Yes	No	Yes	Yes
Arbitrary I/O	No	No	Yes	Yes
Learning Rule	Specified	Specified	Unspecified	Unspecified
Activity Dependent Morphology	No	Some	Yes	Yes

TABLE I

LIST OF ALL THE PROPERTIES OF BIOLOGICAL SYSTEMS THAT ARE INCORPORATED INTO CGPCN OR ARE PRESENT IN ANNS AND NEURAL DEVELOPMENT MODELS.

neurons and branches are considered to be active and their corresponding program is run. The value of the *statefactor* is affected indirectly by CGP programs. The bio-inspiration for the *statefactor* is the fact that not all neurons and/or dendrites branches in the brain are actively involved in each process.

A. *Inputs, Outputs and Information Processing in the Network*

The external inputs (encoding a simulated potential) are applied to the CGPCN and presented to axosynaptic electrical processing chromosomal branches as shown in figure 3. These are distributed in the network in a similar way to the axon branches of neurons. After this the program encoded in the axo-synaptic electrical branch chromosome is executed, and the resulting signal is transferred to its neighbouring active dendrite branches. Similarly we have outputs which read the signal from the CGPCN through dendrite branches. These branches are updated by the axo-synaptic chromosomes of neurons in the same way as other dendrite branches and after five cycles the potentials produced are averaged and this value is used as the external output (see Fig 1). Information processing in the network starts by selecting the list of active neurons in the network and processing them in a random sequence. Each neuron takes the signal from the dendrites by running the electrical processing in dendrites. The signals from dendrites are averaged and applied to the soma program

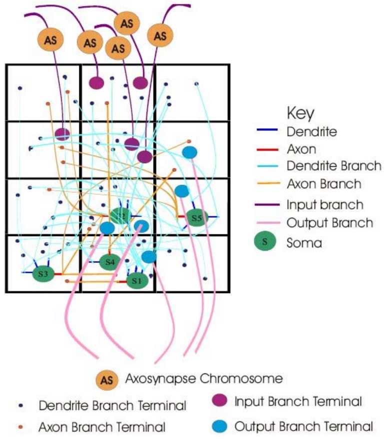


Fig. 3. A schematic illustration of a 3 × 4 CGPCN grid. The grid contains five neurons, each neuron has a number of dendrites with dendrite branches, and an axon with axon branches. Inputs are applied at five random locations in the grid using input axo-synapse branches by running axosynaptic CGP programs. Outputs are taken from five random locations through output dendrite branches. The figure shows the exact locations of neurons and branches as used in most of the experiments as an initial network. Each gird square represents one location, branches and soma are shown spaced for clarity. Each branch location is represented by where its terminal is located. Every location can have as many neurons and branches as the network produces, there is no imposed upper limit.

along with the soma potential. The soma program is run 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 to other neurons through axosynaptic branches. The same process is repeated in all neurons. Description of the seven chromosomes is given in the next section.

B. CGP Model of Neuron

In the model, neural functionality is divided into three major categories: electrical processing, life cycle and weight processing.

Electrical Processing

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

The D chromosome handles the interaction of dendrite branches belonging to a dendrite. It takes active dendrite branch potentials and soma potential as input and the 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 is run (DBL), otherwise it continues processing the other dendrites.

The chromosome responsible for the electrical behaviour of the soma, 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 program encoded in axo-synapse electrical chromosome (AS). The threshold potential of the soma is adjusted to a new value (maximum) if the soma fires.

The potential from the soma is transferred to other neurons through axon branches. The AS program 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 chromosome (WP) is run which updates the *Weights* of neighbouring(branches sharing same grid square) branches. The processed axo-synaptic potential is assigned to the dendrite branch having the *largest* updated *Weight*.

Life Cycle of Neuron

This part is responsible for replication or death of neurons and neurite branches and also the growth and migration of neurite branches. It consists of three life cycle chromosomes responsible for the neuron and 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 CGPCN grid point 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 updated value of the soma *Health* decides whether the soma should produce offspring, should die or continue as it is. If the updated *Health* is above certain threshold it is allowed to produce offspring and if below a certain threshold it is removed from the network along with its neurites. If it produces offspring, then a new neuron is introduced into the network with a random number of neurites at a different random location.

VI. The Game: An Agent plays against an adaptive checkers playing program

The CGPCN checkers player is trained against a minimax based checkers program (MCP). Unfortunately we were unable to adjust the standard of play of the MCP program and it was obviously playing at high level. Each agent in the population plays a game against the MCP with the agent starting each game from a random network. The best playing genotype based on fitness is selected as the parent for the new population and is promoted to the next generation unaltered along with four offspring (mutational variants).

When the experiment starts, the MCP makes the first move and the updated board is the applied to the agent CGPCN. The CGPCN network is then run for five cycles. During this process it updates the potentials of the output (dendrite) branches. These are averaged, and used to decide the direction of movement for the corresponding piece. Each piece is allocated an output dendrite branch in the CGPCN. The potentials of these branches are updated during CGPCN process. The updated values of these potentials are used to decide which piece to move, unless there is a jump, which takes priority. For more than one jump, the piece with highest potential makes the jump.

The game is stopped if either the CGPCN of an agent dies (i.e. all its neurons or neurites dies), or if all its or opponent players are taken, or if the agent or its opponent can not move, or if the alloted number of moves allowed for the game have been taken.

CGP Computational Network (CGPCN) Setup Each CGPCN has neurons and branches located in a 4x4 grid. Initially the number of neurons is 5. Maximum number of dendrites and neurite branches is 5 and 200 respectively. Maximum statefactors are 7 (branch) and 3 (soma). Mutation rate is 5%. Maximum number of nodes per chromosome is 200. Maximum number of moves is 20.

Fitness Calculation

Both the agent and the MCP are allowed to play a limited number of moves and the fitness of the agent is accumulated

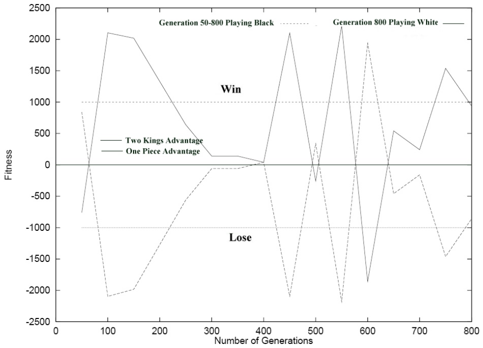


Fig. 4. Graph showing the fitness variation of a well evolved agent against earlier ancestors

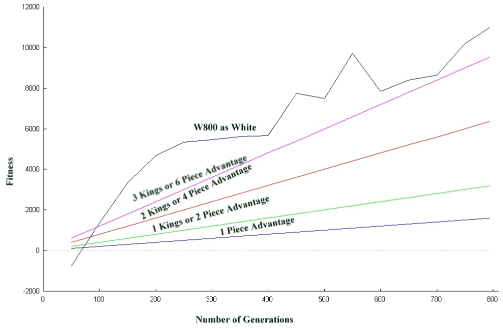


Fig. 5. Graph showing Accumulated fitness variation (right)

at the end this period using the following equation: $Fitness = A + 200N_K + 100N_M - 200N_{OK} - 100N_{OM} + N_{MOV}$, Where N_K represents the number of kings, and N_M represents number of men of the current player. N_{OK} and N_{OM} represent the number of kings and men of the opposing player. N_{MOV} represents the total number of moves 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.

Inputs and outputs of the System

The input 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). The values taken by I are as follows: if empty I=0, if king I=Maximum value(M) $2^{32} - 1$, if piece I=(3/4)M, if opposing piece I=(1/2)M, and finally if opposing king, I=(1/4)M. The board inputs are applied in pairs to all the sixteen locations in the 4x4 CGPCN grid (i.e. two virtual axo-synapse branches in every grid square).

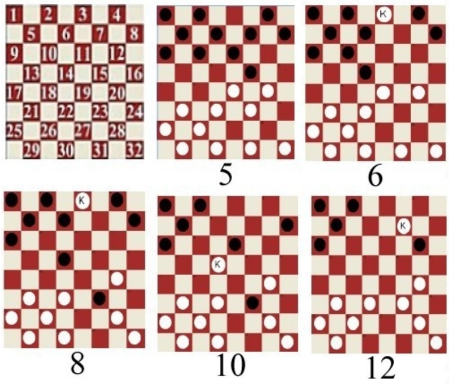
Output is in two forms, one of the outputs is used to select the piece to move and second is used to decide where that piece should move. Each piece on the board has a virtual dendrite branch in the CGPCN. All pieces are assigned a unique ID, representing the CGPCN grid square where its branch is located. Each of these branches has a potential, which is updated during CGPCN processing. The values of potentials determine the possibility of a piece to move (highest potential piece is moved), with jump take priority. If more than one jump then the piece with the highest potential jump, jump with a king take priority (Rules of checkers), unless there are more than one kings. In addition, there are also five virtual dendrite branches distributed at random locations in the CGPCN 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 CGPCN grid.

Results and Analysis During the course of evolution when an agent was evolved against MCP, the agent was never able to beat MCP or even have piece advantage within 20 moves. So it was difficult to assess from the variation in fitness of the agent during the course of evolution and whether it could learn or not. MCP was playing at a higher level, and although the agent learns different moves during the course of evolution, MCP still manages to beat the agent, thus causing its fitness to stay low. The variation in fitness of the agent appears to be random. As the MCP produces a database of game moves, and uses that in the calculation of its next move, it plays different games every time even against the same opponent. Thus it is difficult for evolution to select the best genotype for the next generation. Although the genotype of the best agent is promoted unaltered to the next generation it produces different value of fitness, so it is difficult to obtain any improvement against MCP.

To test whether any learning had taken place we tested well evolved agents against less evolved agents in a one game scenario, and found that the former almost always beats the latter, in some of the cases it ends up in a draw, but even in those cases the well evolved agent ends up with more kings and pieces than the less evolved agent. Thus it is clear that agents are improving their level of play. Figure 4 shows the variation in fitness of a well evolved agent (from generation 800) against the fitness of a series of agents from earlier generations (ancestors from generation 50, 100, 150.....750, 800). It is evident that the well evolved agent playing white always beats the less evolved playing black. Its winning margin is quite variable because its fitness is evaluated over a single game. Figure 5 shows the cumulative fitness of the well evolved agent over its series of games. The figure also shows cumulative fitness variation that would have occurred when a highly evolved agent beat the lesser evolved agent by a fixed margin in every game with various piece advantages. This allows us to assess how the cumulative winning margin varied over evolutionary time.

The table in 6 presents a game between a highly evolved agent and a less evolved agent. The figure also shows some of the important board positions during the game. The highly evolved agent is playing white (generation 1800) and ancestor agent (generation 1050), now playing black.

The 6 moves by each players are sensible, but white ends up with its pieces further advanced. However black leaves an



Move Number	Black	White
1	10 to14	24 to 20
2	6 to 10	23 to 19
3	14 to 23	28 to 19
4	3 to 6	27 to 23
5	12 to 15	19 to 12 12 to 3
6	11 to 14	23 to 19
7	14 to 23	31 to 28
8	10 to 14	3 to 10 10 to 19
9	4 to 7	19 to 14
10	7 to 11	14 to 7
11	23 to 27	30 to 23
12	8 to 12	7 to 16
13	9 to 13	23 to 19
14	2 to 6	21 to 17
15	6 to 11	17 to 10
16	5 to 14 14 to 23	28 to 19
17	11 to 14	19 to 10
18	1 to 5	10 to 1

Fig. 6. Labeled Board and positions at different stages of the game. Numbers beneath boards show the board at moves 5, 6, 8, 10 and 12. Table on left lists the first 18 moves played by the two players

empty square on its back rank after move 4 which later after moving a piece from square 12 to 15 becomes catastrophic one, as it means that white is forced to take two pieces and acquire a king. Reasonable play resumes for a couple of moves but then black moves its piece from square 10 to 14 and allows white's king on the back rank to take two pieces. On move 16 black gains some white pieces but white's piece and king advantage is too great, so white ends up in a winning position.

VII. Conclusion

We have presented a new developmental approach to the construction of neural networks. It is our view, that only through increasing the biological plausibility of neural models will we be able to understand what learning is and how to make artificial systems approach the learning ability exhibited by real brains. As far as we are aware computer checkers is the most substantial problem that developmental neural models have been applied to. However, at this stage we have not demonstrated that we can obtain an extremely good checkers playing program, but we have demonstrated that evolution can create rules that build stable (rather than chaotic or dying) neural systems that play increasingly well. We haven't yet demonstrated that these systems can improve substantially their level of play merely by playing checkers. Perhaps this is not surprising, since for purely computational reasons we were not able to evaluate the quality of each evolved neural program over a long series of games. This is something we intend to do in the future. Our eventual aim is to see if we can evolve a *general* capability for learning.

References

[1] K. Gurney, *An Introduction to Neural Networks*. London: Routledge, 1997.

[2] J. F. Miller and P. Thomson, "Cartesian genetic programming," in *Proc. of the 3rd European Conf. on Genetic Programming (EuroGP)*, vol. LNCS 1802, 2000, pp. 121–132.

[3] C. Panchev, S. Wermter, and H. Chen, "Spike-timing dependent competitive learning of integrate-and-fire neurons with active dendrites," in *Proc. International Conference on Artificial Neural Networks (ICANN)*, J. R. Dorronsoro, Ed., vol. LNCS 2415. Springer-Verlag, 2002, pp. 896–901.

[4] B. Graham, "Multiple forms of activity-dependent plasticity enhance information transfer at a dynamic synapse," in *International Conference on Artificial Neural Networks (ICANN)*, J. R. Dorronsoro, Ed., vol. LNCS 2415. Springer-Verlag, 2002, pp. 45–50.

[5] G. Khan, J. Miller, and D. Halliday, "Coevolution of intelligent agents using cartesian genetic programming," in *Proc. of Genetic and Evolutionary Computation Conference (GECCO)*, 2007, pp. 269 – 276.

[6] J. Smythies, *The Dynamic Neuron*. BradFord, 2002.

[7] E. R. Kandel, J. H. Schwartz, and T. M. Jessell, *Principles of Neural Science, 4th Edition*. McGraw-Hill, 2000.

[8] S. Rose, *The Making of Memory: From Molecules to Mind*. Vintage, 2003.

[9] C. Koch and I. Segev, "The role of single neurons in information processing," *Nature Neuroscience Supplement*, vol. 3, pp. 1171–1177, 2000.

[10] C. Shannon, "Programming a computer for playing chess," *Phil. Mag.*, vol. 41, pp. 256–275, 1950.

[11] R. W. Dimand and M. A. Dimand, *A History of Game Theory: From the Beginnings to 1945*. Urbana: Routledge, 1996, vol. 1.

[12] A. Samuel, "Some studies in machine learning using the game of checkers," *IBM J. Res. Dev.*, vol. 3, no. 3, pp. 210–219, 1959.

[13] J. Schaeffer, *One Jump Ahead: Challenging Human Supremacy in Checkers*. Springer, Berlin, 1996.

[14] D. Moriarty and R. Miikulainen, "Discovering complex othello strategies through evolutionary neural networks," *Connection Science*, vol. 7, no. 3-4, pp. 195–209, 1995.

[15] N. Richards, D. Moriarty, P. McQuesten, and R. Miikkulainen, "Evolving neural networks to play go," pp. 85–96, 1998.

[16] G. Kendall and G. Whitwell, "An evolutionary approach for the tuning of a chess evaluation function using population dynamics," in *IEEE Congress on Evolutionary Computation (CEC 2001)*, 2001, pp. 995–1002.

[17] K. Chellapilla and D. B. Fogel, "Evolving an expert checkers playing program without using human expertise," in *IEEE Trans. on Evolutionary Computation*, vol. 5, no. 5, 2001, pp. 422–428.

[18] J. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural selection*. MIT Press, 1992.

[19] S. Nolfi and D. Parisi, *Genotype for Neural Networks. In Arbib, M.A. ed. Handbook of Brain theory and Neural Networks*. MIT Press, 1995.

[20] A. Cangelosi, S. Nolfi, and D. Parisi, "Cell division and migration in a 'genotype' for neural networks," *Network-Computation in Neural Systems*, vol. 5, pp. 497–515, 1994.

[21] F. Gruau, "Automatic definition of modular neural networks," *Adaptive Behaviour*, vol. 3, pp. 151–183, 1994.

[22] A. Rust, R. Adams, S. George, and H. Bolouri, "Designing development rules for artificial evolution," in *Proceedings of 3rd International Conference on Artificial Neural Networks and Genetic Algorithms (ICANNGA97)*. Springer Verlag, 1997, pp. 509–513.

[23] ——, "Activity-based pruning in developmental artificial neural networks," in *Proceedings of European Conference on Artificial Life (ECAL'97)*, 1997, pp. 224–233.

[24] N. Jakobi, *Harnessing Morphogenesis, Cognitive Science Research Paper 423, COGS*. University of Sussex, 1995.

[25] D. Federici, "Evolving developing spiking neural networks," in *Proceedings of CEC 2005 IEEE Congress on Evolutionary Computation*, 2005, pp. 543–550.

[26] J. C. Astor and C. Adami, "A development model for the evolution of artificial neural networks," *Artificial Life*, vol. 6, pp. 189–218, 2000.

[27] K. L. Downing, "Supplementing evolutionary developmental systems with abstract models of neurogenesis," in *Proc. of the 9th annual conference on Genetic and Evolutionary Computation (GECCO)*. New York: ACM, 2007, pp. 990–996.

[28] J. F. Miller, V. K. Vassilev, and D. Job, "Principles in the evolutionary design of digital circuits-part i. *genetic programming*," vol. 1:1/2, 2000, pp. 7–35.

[29] G. Shepherd, *The synaptic organization of the brain*. Oxford Press, 1990.

[30] A. Van Ooyen and J. Pelt, "Activity-dependent outgrowth of neurons and overshoot phenomena in developing neural networks," *Journal of Theoretical Biology*, vol. 167, pp. 27–43, 1994.