Intelligent agents capable of developing memory of their environment

Gul Muhammad Khan, Julian F. Miller, and David M. Halliday

Electrical Engineering Department, NWFP UET Peshawar, Pakistan,
Electronics Department, University of York, York, YO10 5DD, UK

Abstract. We investigate the emergence of intelligent behaviour of agents in the classic AI learning environment known as Wumpus World. The agent is given a 'brain' that is a new type of developmental neuro-inspired computational network. The neurons are defined by a genotype consisting of seven computational functions that represent electrical and developmental processes inside neuron. The computational functions are evolved using a form of Genetic Programming known as Cartesian Genetic Programming. The 'brain' that occurs by running the genetic programs has a highly dynamic morphology in which neurons grow, and die, and neurite branches together with synaptic connections form and change in response to situations encountered in the external environment. We present results and analyse characteristics of the model and find that the agents appear to exhibit 'instinctive' behaviour and develop a form of memory.

Key words: Generative and developmental approaches, Learning and memory, Artificial Neural Networks

1 Introduction

In this chapter we propose a computational system capable of cognitive and adaptive behaviour inspired by neuroscience. In our view, emergent behaviour in a cognitive system arises as a consequence of the biological development of the brain interacting with an external environment. This process is responsible for discovering the meaning of environmental signals and the creation of mental symbols. We have created a computational brain model inspired by neuroscience in which the brain of an agent develops during environmental interaction in a classic AI problem known as Wumpus World. In this world there are signals that indicate the presence of hazards and rewards. The agent acquires an understanding of the meaning of these signals and their relevance during its lifetime, while its brain is developing.

Recently a number of grand challenges have been proposed in Computer Science both in the US and the UK. In the challenge, The Architecture of Brain and
Mind a number of sub-tasks were identified that would be needed to achieve the higher aim of building a robot with the cognitive capabilities of a human child. One of these is concerned with 'Bottom-up specification, design, and construction of a succession of computational models of brain function'. Our research is concerned with just such a computational model.

Artificial Neural Networks (ANNs), have long been seen as a computational equivalent of the brain, however they have largely ignored many aspects of biological neural systems, particularly neural development. It has been observed that "Mechanisms that build brains are just extensions of those that build the body" [Marcus, 2004]. In addition, it is now known that memory is not a static process and the location and mechanisms responsible for remembered information is in constant (though, largely gradual) change [Rose, 2003]. The physical topology of the neural structures in the brain is constantly changing and is an integral part of its learning capability. Dendrites themselves should no longer be regarded as passive entities that simply collect and pass synaptic inputs to the soma, indeed Koch argues that "Dendritic trees enhance computational power" [Koch and Segev, 2000]. In most cases they shape and integrate these signals in complex ways [Stuart et al., 2001]. Neurons communicate through synapses. Synapses are not simply the point of connection. For a start, there are two types of synapses in the brain: electrical and chemical and synapses can change the strength and shape of the signal over various time scales (see page 241 in [Kandel et al., ]).

In this work we are attempting to obtain a computational analogue of the biological developmental neuron. We have taken the view that, despite the underlying subtlety and complexity of neurons, their gross morphology and connectivity is sufficiently well understood to allow us to identify essential sub-systems and their inputs and outputs.

Since it appears, at present, impossible to design by hand, a computational equivalent of a neuron, we have chosen to use a method of automatic program evolution called Genetic Programming (GP) [Koza, 1992]. In particular, we use a well established and effective form of GP, known as Cartesian Genetic Programming (CGP) [Miller et al., 1997, Walker and Miller, 2008].

In the model [Khan et al., 2007] we have idealized seven neural aspects which we have represented as CGP chromosomes encoding combinational digital circuits. 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. Our collection of chromosomes represent many of the essential aspects of the neuron: soma, dendrites and axon branches, and synaptic connections. The computational network that forms when the seven chromosomes are run continually adapts and changes a network of neurons, neurites and synapses based on its own internal dynamics and external environmental interaction. We have tested and evaluated the capability of the system on the well known artificial intelligence problem known as Wumpus World [Russell and Norvig, 1995].

Section 2 gives a description of CGP. Section 3 reviews previous research on artificial neural development. Section 4 discusses the important biological aspects
that we have included in our model of a computational neuron, and compares and contrasts which features are included in other neural systems (both biological and artificial). Section 5 describes the new computational model at the network level. Section 6 describes in detail the CGP neuron and gives the algorithms that define how the whole network changes and develops. Section 7 discusses how information is processed in the model. In section 8 we describe how we applied our computational model to the Wumpus World problem and give our results and analysis. Finally, in section 9 we conclude the chapter and describe future work.

2 Cartesian Genetic Programming (CGP)

Cartesian Genetic Programming [Miller et al., 1997, Miller and Thomson, 2000] was developed from the work of Miller and Thomson for the evolutionary design of feed forward digital circuits. In CGP, programs are represented by directed acyclic graphs. Graphs have advantages in that they allow implicit re-use of subgraphs. In its original form CGP used a rectangular grid of computational nodes (in which nodes were not allowed to take their inputs from a node in the same column). However, later work relaxed this restriction by always choosing the number of rows to be one (as used in this work). The genotype in CGP has a fixed length. The genes are integers which encode the function and connections of each node in the directed graph. However, the phenotype is obtained via following referenced links in the graph and this can mean that some genes are not referenced in the path from program inputs to outputs. This results in a bounded phenotype of variable length. As a consequence there can be non-coding genes that have no influence on the phenotype, leading to a neutral effect on genotype fitness. The characteristics of this type genotypic redundancy have been investigated in detail and found to be extremely beneficial to the evolutionary process on the problems studied [Miller et al., 2000, Vassilev and Miller, 2000, Yu and Miller, 2001]. Indeed, it has been shown that evolutionary search proceeds most quickly when extraordinary levels of redundancy are present [Miller and Smith, 2006] (i.e. when 95% of all genes are redundant).

Each node in the directed graph represents a particular function and is encoded by a number of genes. The first gene encodes the function that the node represents, and the remaining genes encode where the node takes its inputs from. The nodes take their inputs from either the output of a previous node or from a program input (terminal). The number of inputs that a node has is dictated by the number of inputs that are required by the function it represents. The general 2D form of a CGP program is shown in figure 1.
Fig. 1. General form of two-dimensional CGP. It is a grid of nodes whose functions are chosen from a set of primitive functions. Each node is assumed to take as many inputs as the maximum function arity \( a \). Every data input and node output are labeled consecutively (starting at 0) which gives it a unique data address which specifies where the input data or node output value can be accessed (shown in the figure on outputs of inputs and nodes). Nodes in columns cannot be connected to each other. In most cases the graph is directed (as in this paper) so that a node may only have its inputs connected to either input data or the output of a node in a previous column. In general there may be a number of output genes \( (O_i) \) which specify where the program outputs are taken from. The structure of the genotype is seen below the schematic. All node functions genes \( f_i \) are integer addresses in a look-up table of functions. All connection genes \( C_{ij} \) are integers taking values between 0 and the address of the node at the bottom of the previous column of nodes.
3 Neural Development

Artificial neural networks are intended to mimic, in some sense, the computational models of nervous systems. However, many ANN models ignore the fact that the neurons in the nervous system are part of the phenotype which is derived from the genotype through a process called development [Kumar, 2003]. The information specified in the genotype determines some aspects of nervous system, which specifies the rules of developing the nervous system based on environmental interaction during developmental phase. Natural organisms, however, not only possess nervous systems but also genetic information stored in the nucleus of their cells (genotype).

Development schemes are intended to increase the scalability of ANNs by having a minimum number of genes defining the properties of the network, instead of having a one to one relationship between genotype and phenotype.

Specific groups of genes can influence several otherwise unrelated phenotypic traits, so that the dimension of the genotype can be quite independent of the phenotypic size. For example, it is estimated that there are only 30–40 thousands genes in the human genotype (45 million DNA bases out of a total $10^9$), while $10^{14}$ cells constitute a mature phenotype [Elliot and Elliot, 2001], [Lodish et al., 2003].

Parisi and Nolfi argued that, if neural networks are viewed in the biological context of artificial life, they should be accompanied by genotypes which are part of a population and inherited from parents to offspring [Parisi, 1997], [Parisi and Nolfi, 2001].

In their work they used a growing encoding scheme to evolve the architecture and the connection strengths of neural networks [Nolfi and Parisi, 1995]. They used their network to control a small mobile robot (for a similar method see [Husbands et al., 1994]). The network was defined in a 2-D space, where a collection of artificial neurons are distributed with growing and branching axons. The genetic code inside them specifies the instructions for axonal growth and branching in neurons. Connections between neurons are made when an axon of neuron reaches other neuron. The way the axons grew was dependent on the strengths of signals passing through the network, this meant that different neural networks arose from the same genetic code in different external environments.

Cangelosi proposed a neural development model, which starts with a single cell undergoing a process of cell division and migration [Cangelosi et al., 1994]. Each cell produces two daughter cells with new cells separated in the 2-dimensional space. This cell division and migration continues until a collection of neurons arranged in 2D space is developed. At the end neurons grow their axons to produce connection among each other. This process continues until a neural network is developed. The rules for cell division and migration are stored in genotype, for a related approach see [Dalaert and Beer, 1994].

Gruau proposed a graph re-writing method [Gruau, 1994]. His network starts with a single cell which undergoes various stages of cell division and differentiation until a complete neural network is produced. Each cell is divided into two daughter cells, with some new connections being introduced and old connections...
strengthened. The rules for the process of cell division and transformation are specified in the genotype. The genotype used in Gruau’s model is the form of a binary tree structure as in GP [Koza, 1992]. The genotype tree starts from the top node which is the initial cell. Each node of the genotype in the tree encodes the operation of that cell and the two sub-trees specify the operation that should be applied to the two daughter cells. The neural network is developed by following the tree using instructions in these cells. The network ends up at terminal cells having no sub-trees.

In subsequent work, Gruau introduced a genotype-phenotype mapping that allowed the repetition of the phenotypic structure by re-using the same genetic information. In this case the terminal cells (nodes) point to the other trees. This encoding method can produce complex phenotypic networks from a compact genotype. Gruau called this method “automatic definition of neural sub networks (ADNS)” [Gruau, 1994].

Karl Sims used a graph based GP approach to evolve virtual robotic creatures. The morphology of these creatures and the neural systems for controlling their muscle forces were both genetically determined [Sims, 1994]. The genotypes were structured as directed graphs of nodes and connections. When a creature is synthesized from its genetic description, the neural components described within each part are generated along with the morphological structure.

Rust and Adams devised a developmental model coupled with a genetic algorithm to evolve parameters that grow into artificial neurons with biologically-realistic morphologies [Rust et al., 2000], [Rust and Adams, 1999]. They also investigated activity dependent mechanisms [Rust et al., 1997] so that neural activity would influence growing morphologies. Although Rust and Adams 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.

Quartz and Sejnowski [Quartz and Sejnowski, 1997] have laid down a powerful manifesto for the importance of dynamic neural growth mechanisms in cognitive development and Marcus has emphasized the importance of growing neural structures using a developmental approach “I want to build neural networks that grow, networks that show a good degree of self-organization even in the absence of experience” [Marcus, 2001].

Jakobi created an impressive artificial genomic regulatory network, where genes code for proteins and proteins activate (or suppress) genes [Jakobi, 1995]. He used the proteins to define neurons with excitatory or inhibitory dendrites. The individual cell divides and moves due to protein interactions with an artificial genome, causing a complete multicellular network to develop. After differentiation each cell grows dendrites following chemical sensitive growth cones to form connections between cells. This develops a complete recurrent ANN, which is used to control a simulated Khepera robot for obstacle avoidance and corridor following. Figure 2 shows a well evolved model of ANN obtained by Jakobi for both corridor following and obstacle avoidance. During every generation genotypes develop phenotypical structures, which are tested, and the best genotypes
Intelligent agents capable of developing memory of their environment are selected for breeding. Artificial evolutionary operators such as crossover and mutation are used to create offspring genotypes.

A number of researchers have studied the potential of Lindenmeyer systems [Lindenmeyer, 1968] for developing artificial neural networks and generative design. Boers and Kuiper have adapted L-systems to develop the architecture of artificial neural networks (ANNs) (numbers of neurons and their connections) [Boers and Kuiper, 1992]. They evolved the rules of an L-system that generated feed-forward neural networks. They found that this method produced more modular neural networks that performed better than networks with a pre-defined structure. Hornby and Pollack evolved L-systems to construct complex robot morphologies and neural controllers [Hornby and Pollack, 2001].

Federici presented an indirect encoding scheme for development of a neuro-controller, and compared it with a direct scheme [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. Federici used two steps to produced the neuro-controllers:

- A growth program in a genotype to develop the whole multi-cellular network in the form of a phenotype. The growth program inside each cell is based on local variables and implemented by a simple recursive neural network without hidden layer (Similar to our use of CGP).
- In a second step it translates all the cells into spiking neurons.

Each cell of a mature phenotype is a neuron of a spiking neuro-controller. The type and metabolic concentrations of a cell are used to specify the internal dynamics and synaptic properties of its corresponding neuron. The position of the cell within the organism is used to produce the topological properties of a neuron: its connections to inputs, outputs and other neurons.
The network was implemented on a kephera robot and its performance was tested with direct and indirect encoding schemes. The indirect method reached the high fitness faster than the direct one, but had difficulty in refining the final fitness value.

Downing favours a higher abstraction level in neural development to avoid the complexities of axonal and dendritic growth while maintaining key aspects of cell signalling, competition and cooperation of neural topologies in nature [Downing, 2007]. He developed a system and tested it on a simple movement control problem known as starfish. The task for the k-limbed animate is to move away from its starting point as far as possible in a limited time, producing encouraging preliminary results.

4 Key features and biological basis for the CGPCN model

This section draws together the key ideas from biology, Artificial Neural Networks (ANNs), neural development, and describes the main features of CGPCN model, along with the biological basis of these ideas. Table 1 lists all the properties of biological systems that are incorporated into CGPCN. Table 1 also shows the presence and absence of these properties in existing ANNs and neural development models [Kumar, 2003].

Table 1 lists the important properties of both biological and artificial neural systems. We discuss the meaning of each property below.

**Neuron Structure:** Biological neurons have dendrites and axon with branches. Each neuron has a single axon with a variable number of axon branches. It also has a variable number of dendrites and dendrite branches. Some published developmental neural models have these features. In our model neurons have three key morphological components:
- Dendrites with branches that receive and process inputs.
- Cell body which processes signals from dendrites.
- An axon which transfers signals to other neurons through axon branches.

ANNs have only nodes and connections, there is no concept of branches.

**Interaction of Branches:** In biological dendrites the signals from different branches actually interact with each other, whereas in ANNs and published neural development there is no concept of interaction between connections. We have adopted this feature and evolved the function for interaction between branches, as no precise mathematical model is available to approximate this function. Although ANNs and neural development sometimes regard connections between neurons as dendritic, they are far from biological dendrites in the types of morphology that can exist [Kandel et al., 2000].

**Neural function:** Biological neurons are decision making entities, they integrate the signal and fire. Some ANN models have adopted this in the form of spiking neurons. We also do this and allow our neurons to integrate the signal received from branches and fire an action potential.
Intelligent agents capable of developing memory of their environment.

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 1. List of all the properties of biological systems that are incorporated into CGPCN or are present in ANNs and neural development models.

**Resistance:** Branches in biological neurons have the property of electrical resistance. Resistance not only affect the strength of the signal but also is related to the length of the branch. There is no concept of branch resistance in the ANN literature, but there is a concept of length in some of the neural developmental models. In our model neurite branches have a quantity which we call resistance which is related to their length and which affects the strength of the signal that is transmitted. Resistance also plays an important role in branch growth during the developmental phase.

**Health:** Biological neurons have property which one can loosely consider to be ‘Health’. Since some neurons are weak and die, or may be diseased in some way. Generally speaking, biological neurons that are inactive for a long time tend to dwindle away. There is no concept of health of neurons or branches in ANNs.
or neural development models. However, we have adopted this property to allow the replication or death of neurons and neurites. Health also affects the signal processing inside neurons and neurites.

**Neural Activity**: Neural activity refers to the degree to which neurons and neurites are active. In biology, not all neurons and neurites actively participate in every function that brain performs. In ANNs and neural development all neurons have to be processed before the function of the network can be assessed. This means that all the neurons are always active in ANNs. Following biology, in our model the network dynamics decides about which neuron or neurite should be processed for a particular task. This is done by either making it active or changing the time for which it will be inactive.

**Synaptic Communication**: Electrical signals are transferred from one neuron to another through synapses. Synapses are not just the point of contact, as considered in many published models of ANNs and neural development, they modulate the signal and provide a complex mechanism for signal transfer. We have evolved CGP programs to find the useful mechanisms to allow signal transfer across a synapse. Synapses in biological systems affect the potentials of the branches nearby through changes in the concentrations of chemical (i.e. ions in space between neurons). Although synapses in the CGPCN are not chemical, when a synapse in the CGPCN is made it causes the weights and potential values of the neighbouring branches to update. This is analogous to the chemical changes at synapses.

**Arrangement of Neurons**: The overall architecture of both ANNs and many neural developmental systems is fixed once developed, whereas biological neurons exist in space, interact with each other and move their branches from one place to other. We have adopted similar mechanism by placing neuron in Euclidean grid space, such that they have spatial proximity and this affects how they interact with each other. The axon and dendrite branches are also allowed to navigate over the Euclidean grid. This means that the morphology of the network is able to change during problem solving.

**Spiking (Information processing)**: Signals are passed from neuron to neuron via spikes (nerve impulses) in biological brain, some of the ANN models (Spiking Neural networks) and Neural development models used spiking mechanism for signal processing. We have also adopted a similar mechanism in our model, as signals are transferred to other neurons only if the neuron fires.

**Synaptic Plasticity**: A key concept in understanding neural systems is plasticity. Plasticity means the ability to change or reform. Plasticity occurs at several levels in the nervous system. Much of the dynamic capabilities exhibited by neural systems results from synaptic plasticity, which refers to changes in synaptic transmission [Debanne et al., 2003]. This synaptic plasticity occurs at both post synaptic and pre-synaptic levels. It can involve changes in the postsynaptic excitability (the probability of generating action potential in response to a fixed stimulus), which depends on the previous pattern of input [Gaiarsa et al., 2002]. The numbers of receptors (sites of neurotransmitter action) on the membrane can also be altered by synaptic activity [Frey and Morris, 1997]. These processes...
Intelligent agents capable of developing memory of their environment can interact resulting in positive feedback effects, where some cells never fire and others may saturate at some maximal firing rate.

Synaptic plasticity is incorporated in the CGPCN by introducing three types of weights: 1) dendrite branch, 2) soma, and 3) axon branch.

These weights can be adjusted by genetic processes during development of the network. Changes in the dendrite branch weight are analogous to the amplifications of a signal along the dendrite branch (see [London and Husser, 2005]), whereas changes in the axon branch (or axo-synaptic) weight are analogous to changes at the pre-synaptic level and postsynaptic level (at synapse). Inclusion of a soma weight is justified by the observation that a fixed stimulus generates different responses in different neurones [Traub, 1977].

Synaptic plasticity in SNN models is primarily based on the work of Hebb [Hebb, 1949], the basic tenet of which is that repeated and persistent stimulation of a postsynaptic neuron by a pre-synaptic neuron results in a strengthening of the connection between the two cells. A recent development of this idea is the use of spike time dependent plasticity (STDP) rules for updating weights in SNN networks [Roberts and Bell, 2002], [Song et al., 2000]. Weight changes depend on the relative timing of pre- and postsynaptic spikes. This process has been documented in a range of neural systems, and a number of rules have been proposed. Weights can either increase, decrease or remain unchanged, such methods are reviewed in [Roberts and Bell, 2002]. Although these schemes use an update rule depending on the firing time of only two neurons, they can result in interesting patterns of global behavior, including competition between synapses [Van Rossum et al., 2000].

We adopt aspects of the above ideas in our CGPCN system by allowing the three different types of weight to change in response to firing of neurons. After every neural firing, its life cycle and weight processing chromosomes are run, this updates the soma weight and health (See later for details in CGP Neuron sections). These weight changes allow neurons and branches to become more active when they are involved in processing data. As explained in the next section, dendrite and axon branches, and soma which are more active are updated more frequently as their genetic code is run only when they are active. This interaction allows the patterns of activity propagating through the network to influence developmental and evolutionary processes. This interaction is known to be important for living neural systems, where sensory input and other environmental factors are important in shaping developmental aspects (see page 1125 onwards in [Kandel et al., 2000]).

Plasticity is usually imposed on ANN models, whereas in biological brains is an emergent consequence of underlying processes. We have adopted this by allowing the network to develop its own plasticity, instead of imposing it.

**Developmental Plasticity:** 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 processes of the brain is affected a lot by external environmental signals. This phenomenon is called Developmental Plasticity. One form of developmental plasticity is synaptic pruning...
Intelligent agents that develop a memory of an environment [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 rules explained in detail in section 6. An initial random connectivity pattern is used to avoid evolution spending extra time in finding connections in the early phase of neural development.

**Arbitrary I/O:** Biological neural systems can adapt to the change in number of sensory inputs. In almost all ANN and neural developmental models the number of inputs and outputs is predetermined and fixed. However, architecture of our network changes all the time, so new input/output neurons can be introduced into CGPCN at runtime without affecting the overall operation of the network.

**Learning Rule:** The learning rule is imposed on the ANNs and neural development systems. In the model we proposed we do not know what would be the best learning rules to use so we allow these to be discovered through evolution.

**Activity Dependent Morphology:** There are few proposed models in which changes levels of activity (in potentials or signals) between neurons leads to changes in neural morphology. This can occur in the model we propose.

In summary, we have idealized the behaviour of a neuron in terms of seven main processes. One to three are electrical processes, four to six are lifecycle mechanisms, seven is weight processing:

1. Local interaction among neighbouring branches of the same dendrite.
2. Processing of signals received from dendrites at the soma, and deciding whether to fire an action potential.
3. Synaptic connections which transfer potential through axon branches to the neighbouring dendrite branches.
6. Creation or destruction of neurons.
7. Updating the synaptic weights (and consequently the capability to make synaptic connections) between axon branches and neighbouring dendrite branches.

Each aspect is represented with a separate chromosome (CGP program).

In the next section we will describe the model of CGP Computational Network, its sub-parts, evolutionary strategy, and its interfacing with the external environment.

5 The CGP Computational Network (CGPCN)

The CGPCN network has two main aspects:

a) Neurons with a number of dendrites, with each dendrite having a number of branches and an axon having a number of axon branches.

b) A genotype representing the genetic code of the neurons. Each genotype consists of seven chromosomes, each represented as a digital circuit.

The first aspect (a) defines mainly the neural components and their properties, and the second (b) is concerned with the internal behaviour of the neurons in the network. These chromosomes represent the functionality of different parts of the neuron. During evolution the second aspect (genotype) is evolved towards the best functionality, whereas the first aspect (the neural components and their properties) only changes during the lifetime of the network, i.e while it is performing the learning task. The CGPCN has been organized in such a way that neurons are placed randomly in a two dimensional grid (the CGPCN grid) so that they are only aware of their spatial neighbours (as shown in figure 3). The initial number of neurons is specified by the user. 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 during this process, branches may grow or shrink and move from one CGPCN grid point to another. They can produce new branches and can disappear, neurons may die or produce new neurons. Axon branches transfer information only to dendrite branches in their proximity. This process is performed by passing the signals from all the neighbouring branches through a CGP program, acting as an electro-chemical synapse, which updates the values of potential only in neighbouring branches. Electrical potential is used for internal processing of neurons and communication between neurons. It is an integer. Inputs and outputs are also converted in term of potentials (integers) before being applied to the network.

In the next four subsections we will describe the basic variables of CGPCN (Resistance, Health, Weight and Statefactor). The genotype representation in the form of Cartesian Genetic programs, the Evolutionary Strategy, and the way inputs and outputs are applied to the network.
Fig. 3. 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. Three are electrical and process a simulated potential in the dendrite (D), soma (S) and axo-synapse branch (AS). Three more are developmental in nature and are responsible for the life-cycle of neural components (shown in grey). They decide whether dendrite branches (DBL), soma (SL) and axo-synaptic branches (ASL) 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.
5.1 Health, Resistance, Weight and Statefactor

Four variables are incorporated into the CGPCN, representing either the fundamental properties of the neurons (*health*, *resistance*, *weight*) or as an aid to computational efficiency (*statefactor*).

Each dendrite branch and axo-synaptic connection has three variables assigned to them. These variable are called *health*, *resistance* and *weight*. The values of these variables are adjusted by the CGP programs (see below). The *health* variable is used to govern replication and/or death of dendrites and axon branches. The *resistance* variable controls growth and/or shrinkage of dendrites and axon branches. The biological basis for the *weight* variable was discussed previously. This variable is used in calculating the potentials in the network. Each soma has only two variables: *health* and *weight*. The use of these variables is summarised in figure 11. *Health*, *weight* and *resistance* are represented as integers.

A *statefactor* is used as a parameter to reduce computational burden, by keeping some of the neurons and branches inactive for a number of cycles. When the *statefactor* is zero the neurons and branches are considered to be active and their corresponding program is run. The value of the *statefactor* is affected by CGP programs, as it is dependent on the outputs of the CGP electrical processing chromosomes (see later).

5.2 Cartesian Genetic Program (Chromosome)

The CGP function nodes used here consist of multiplexer-like operations. Each function node has three inputs and implements one of the four functions as shown in figure 4 [Miller et al., 2000]. We have used a one dimensional geometry for the work we describe, in which the number of rows (see figure 1) is chosen to be 1.

\[ F_1 = a \cdot r + b \cdot c; \]
\[ F_2 = a \cdot r + b \cdot c; \]
\[ F_3 = a \cdot c + b \cdot c; \]
\[ F_4 = a \cdot c + b \cdot c; \]

*Fig. 4.* Multiplexer diagram, showing inputs A, B and C, and function \( F_i \). The figure also lists all the four multiplexer functions that can be used.

Here a, b and c are the inputs to the node (as shown in Figure 4). These functions are arithmetic with operations additions (+), multiplications (.) and inversion (maximum number minus the number). All the inputs and outputs are 8-bit integers.
Fig. 5. 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 type in the genotype is indicated by an underlined gene. 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.)
Figure 5 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.

Here the number of rows used is one as described earlier. Thus the number of columns is equal to number of nodes. The maximum number of nodes are defined by the user. These nodes are not necessarily all connected. Inputs are applied to CGP chromosomes in two ways:

- Scalar
- Vector

In the former, the inputs and outputs are integers while in the latter inputs required by the chromosome are arranged in the form of an array, which is then divided into 10 CGP input vectors. If the total number of inputs can’t be divided into ten equal parts, then they are padded with zeros. This allows us to process an arbitrary number of inputs by the CGP circuit chromosome simply by clocking through the elements of the vectors. In general CGP can’t take variable number of inputs, so this method allows it to take variable number of inputs at run time. As the inputs are arranged in the form of vectors, and each vector can have arbitrary number of elements. This method adds some noise and this is more pronounced when the number of inputs are less than ten, as we pad it with zero when the number of inputs can’t be divided into ten sub vectors. But as the number of inputs increases the effect that this has is reduced.

5.3 Evolutionary Strategy

The evolutionary strategy utilized is of the form $1 + \lambda$, with $\lambda$ set to 4, i.e. one parent with 4 offspring (population size 5) [Yu and Miller, 2001]. The parent, or elite, is preserved unaltered, whilst the offspring are generated by mutation of the parent. The best chromosome is always promoted to the next generation, however, if two or more chromosomes achieve the same highest fitness then newest (genetically) is always chosen [Miller et al., 2000].

The steps in the evolutionary cycle are as follows:

- Create a random population of five genotypes (Each genotype consists of seven chromosomes of neuron)
- Create a CGPCN with random number of dendrites and branch structures.
- An evolutionary generation consists of:
  - For each genotype $C$ in the population:
    - Produce a copy of the random CGPCN
    - Run the genotype on the CGPCN.
    - Calculate fitness of the resulting CGPCN, $F(C)$
  - From population select best $F(C)$; if two or more are equally best then pick newest of them [Miller et al., 2000]
- Create new population by mutation, keeping the promoted genotype unchanged.
- Continue until either a maximum number of generations, or a solution is found.

Thus, network grow and develop through executing the programs encoded in the genotype. We always start from the same initial random network of neurons, dendrites, and dendrite and axon branches.

The promoted genotype is mutated to produce four new genotypes (the offspring) in the following way:

- Calculate the number of genes to be mutated i.e.
  \[ \text{number of genes to be mutated} = \text{number of genes} \times \text{mutation rate/100}. \]
  Where
  \[ \text{number of genes} = (\text{number of inputs per node} + 1) \times \text{number of nodes per chromosome} \times \text{number of chromosomes} \]
- The number of inputs per node is 3 and the number of chromosomes per genotype is 7.
- Select pseudo-randomly one gene at a time and mutate it pseudo-randomly.

Mutation of a gene means:
- if it is a connection then replace with another valid connection.
- if it is a function then replace with another valid function.

5.4 Inputs and Outputs

The inputs are applied to the CGPCN through axon branches by using the axo-synaptic electrical processing chromosomes. These branches are distributed in the network in a similar way to the axon branches of neurons as shown in figure 6. These branches could be regarded as 'input neurons'. They take the input from the environment and transfer it directly to input axo-synapse. When inputs are applied to the system, 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 output neurons which read the signal from the system through output dendrite branches. These output dendrite branches are distributed across the network as shown in Figure 6. These branches are updated by the axo-synaptic chromosomes of neurons in the same way as other dendrite branches.

The output from the output neuron is taken without further processing after every five cycles. The number of inputs and outputs can change at run time (during development), a new branch for input or output can be introduced into the network or an existing branch can be removed. This allows CGPCN to handle arbitrary number of inputs and outputs at run time.

In the next section we will describe the complete neuron model along with its sub-processes.
Intelligent agents capable of developing memory of their environment

Fig. 6. A schematic illustration of a $3 \times 4$ 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 axo-synaptic CGP programs. Outputs are taken from five random locations through output dendrite branches. The figure shows the exact locations of neurons and branches that were used for the initial network in most of the experiments. Each grid square represents one location, branches and soma are shown spaced for clarity. Every location can have as many neurons and branches as the network produces, there is no upper limit.
Fig. 7. Electrical processing in neuron at different stages, from left to right branch potentials are processed by the dendrite electrical program D, then averaged at each dendrite, and soma, which processes it further using the soma program S, giving a final soma potential. This is fed in to a comparator which decides whether to fire an action potential. This is transferred using the axo-synapse program AS.

6 CGP Model of Neuron

In the model neural functionality is divided into three major categories.

- Electrical Processing
- Life Cycle
- Weight Processing

These categories are explained in detail one by one in the subsections below.

6.1 Electrical Processing

The electrical processing part is responsible for signal processing inside neurons and communication between neurons. It consists of following three chromosomes (as shown in Figure 8):

- Electrical Processing in dendrite
- Electrical Processing in soma, and
- Electrical Processing in axo-synaptic branch

The way they process electrical signal and transfer to other neurons is shown in Figure 7.
Electrical Processing in Dendrite This is a vector processing chromosome. This chromosome handles the interaction between potentials of different dendrite branches belonging to the same dendrite. Figure 8 shows the inputs and outputs. The input consists of potentials of all the active branches connected to the dendrite and the soma potential. Since in practice there are many dendrite branch potentials and one soma potential, we increase the importance of the soma potential by creating multiple entries (equal to number of active dendrite branches) of it (in the input vector) before applying it to the CGP program encoded in the chromosome. The CGP program produces the new values of the dendrite branch potentials as output. Figure 8 shows the inputs and outputs to Dendrite Electrical CGP (D). Subsequently the potential of each branch is then processed by adding weighted values of Resistance, Health, and Weight using following equation.

\[ P = (\hat{P} + \alpha_D H + \beta_D W - \gamma_D R) \& \text{Mask} \] ........................ (4.1)

Where \( P \) and \( \hat{P} \) are the potential and updated potential respectively. H, W and R are the health, weight and resistance of dendrite branch respectively. \( \alpha_D, \beta_D \) and \( \gamma_D \) are the adjustment parameters having values between 0 and 1, in this case they are 0.02, 0.05 and 0.05 respectively. The processed potential is masked to avoid overflow, based on how many bits are used for processing. The above equation shows that as the health and weight of the branch goes up so does its potential, and as the resistance goes up the potential goes down (usual resistive behavior). Increase in health cause an increase in potential is justified by the fact that healthy branch facilitates the flow of potential. Weights are responsible
Intelligent agents that develop a memory of an environment for amplification of potential, thus high value of weight cause an increase in potential.

The Statefactor of branches is adjusted based on the updated value of branch potential. The branch is made more active if the change in potential after processing with D increases and vice versa. This is done to encourage more sensitive branches to participate in processing by keeping them active. We set up a range of thresholds for assigning the statefactor. If any of the branches are active (has its statefactor equal to zero), its life cycle CGP program (DBL) is run, otherwise it continues processing the other dendrites. The same process is repeated for all the dendrites and their corresponding branches. After processing all the dendrites, the average value of potentials of all the dendrites is taken which in turn is the average value of all the active dendrite branches attached to them. This potential and the soma potential are applied as inputs to the CGP soma electrical processing chromosome as described below.

**Electrical Processing in Soma** This is a scalar processing chromosome. This chromosome is responsible for determining the final value of soma potential after receiving signals from all the dendrites. The potentials of all dendrites are averaged, which in turn are the average of potentials of active branches attached to them as shown in figure 7. This average potential along with the soma potential is applied as input to the soma electrical processing chromosome (S) as shown in Figure 8.

The chromosome produces an updated value of the soma potential ($\hat{P}$) as output, which is further processed with a weighted summation of Health ($H$) and Weight ($W$) of the soma using following equation.

$$P = (\hat{P} + \alpha_S H + \beta_S W) \& \text{Mask}$$

Where $\alpha_S$ and $\beta_S$ have been chosen with the values 0.02 and 0.05.

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 the soma fires it is kept inactive (refractory period) for a few cycles by changing its statefactor. After this the soma life cycle chromosome is run, and the firing potential is sent to the other neurons by running the axo-synapse electrical processing chromosome. The threshold potential of soma is also adjusted to a new value (maximum) if the soma fires. If soma does not fire, the value of processed potential is checked, and the following actions are taken:

- If it is below one third of the maximum value, its statefactor is set to a higher value, so that it is kept inactive for three cycles. Which means that if the potential of soma is a low value so that does not fire, the soma is kept inactive for more time. Only soma which fire are encouraged and kept more active.

- If its value is above one third and below half then it is kept inactive for just one cycle.

- If it is higher than half the maximum value it is kept active for next cycle and its life cycle is run. This means that if a soma has a high potential it is kept active for the next cycle.
Electrical Processing in Axo-Synaptic Branch  This is a vector processing chromosome. The potential from the soma is transferred to other neurons through axon branches. Both the axon branch and the synapse are considered as a single entity with combined properties. Axo-synapses transfer the signal only to the neighbouring active dendrite branches as shown in figure 9. Branches sharing the same grid square form a neighbourhood. Figure 8 shows the inputs and outputs to the chromosome responsible for the electrical processing in each axo-synaptic branch. As mentioned before, the soma potential is biased (by introducing multiple entries of soma potential to increase its impact).

The chromosome produces the updated values of the neighbouring dendrite branch potentials and the axo-synaptic potential as output. The axo-synaptic potential is then processed as a weighted summation of Health, Weight and Resistance of the axon branch using the following equation.

$$ P = (\tilde{P} + \alpha_{AS} H + \beta_{AS} W - \gamma_{AS} R) \& \text{Mask} \qquad (4.3) $$

Where $P$ and $\tilde{P}$ are the potential and updated potential respectively. H, W and R are the health, weight and resistance of axon branch respectively. $\alpha_{AS}$, $\beta_{AS}$ and $\gamma_{AS}$ are the adjustment parameters having values between 0 and 1, in this case they are 0.02, 0.05 and 0.05 respectively.

The axo-synaptic branch weight processing program (see figure 10) is run after the above process and the processed axo-synaptic potential is assigned to the dendrite branch having the highest updated Weight. The Statefactor of branches is adjusted based on the updated value of branch potential. The branch is made more active if the change in potential during Axo-Synapse Electrical CGP (AS) process is more and vice versa. We set up a range of thresholds for assigning the statefactor (see section 5). If any of the branches are active (has its statefactor equal to zero), its life cycle CGP program is run, otherwise it
continues processing the other axon branches. The axo-synaptic branch CGP is run in all the active axon branches one by one.

**Fig. 10.** Weight processing in axo-synaptic branch with its corresponding inputs and outputs

6.2 Weight Processing

This is a vector processing chromosome. Weight processing is responsible for updating the weights of branches. It consists of one chromosome. The weights of axon and dendrite branches affect their capability to modulate and transfer the information (potential) efficiently. Weights affect almost all the neural processes either by virtue of being an input to a chromosomal program or as a factor in post processing of signals.

Figure 10 shows the inputs and the outputs to the axo-synaptic weight processing chromosome. The CGP program encoded in this chromosome takes as input the weight of the axo-synapse and the neighbouring (same CGPCN grid square) dendrite branches and produces their updated values as output. The synaptic potential produced at AS is assigned to the dendrite branch having the highest weight after weight processing.

6.3 Life Cycle of Neuron

This part is responsible for growth or decrease in the number of neurons and neurite branches and also the growth and migration of neurite branches. It consists of three chromosomes:

- Life Cycle of dendrite branch
- Life Cycle of soma
- Life Cycle of axo-synapse branch
**Life Cycle of Dendrite Branch** This is a scalar processing chromosome. Figure 11 shows the inputs and outputs of the chromosome. This process updates *resistance* and *health* of the branch. Variation in *resistance* of a dendrite branch is used to decide whether it will grow, shrink, or stay at its current location. If the variation in *resistance* during the process is above certain threshold ($R_{DB}$) the branch is allowed to migrate to a different neighbouring location at random. The neighbouring location can be one of the eight possible squares around a rectangular grid. The branch can move to either of the eight neighbouring squares at random. It can move only one square away at a time. Change in resistance can be negative (shrinkage) or positive (growth). In both cases, the absolute change in resistance is used to decide if the branch should move from its current grid square to another grid square. Growth and shrinkage do not occur within one grid square as the square represents a single location. Growth and shrinkage is identified by increases or decreases in resistance during the process.

The updated value of dendrite 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 certain threshold ($H_{db_{max}}$) it is allowed to produce offspring and if below certain threshold ($H_{db_{min}}$), it is removed from the dendrite. Producing offspring results in a new branch at the same CGPCN grid point connected to the same dendrite.

The values of ($R_{DB}$), ($H_{db_{max}}$) and ($H_{db_{min}}$) are specified by the user.

**Life Cycle of Soma** This is a scalar processing chromosome. Figure 11 shows inputs and outputs of the soma life cycle chromosome. This chromosome is intended to evaluate the life cycle of a neuron. This 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 a certain threshold ($H_{s_{max}}$) it is allowed to produce offspring and if below a certain threshold, ($H_{s_{min}}$) it is removed from the network along with its dendrites, dendrite and axon branches. If it produces
offspring, then a new neuron is introduced into the network with a random num-
ber of dendrites and, dendrite and axon branches. This random number has an upper \((B_{\text{max}})\) and lower \((B_{\text{min}})\) limit, in all our experiments we chose \((B_{\text{max}})\) as 5 and \((B_{\text{min}})\) as 2 for the number of dendrites and branches. This new neuron is placed at a pseudo random grid location (square). Soma and branches are pro-
vided with an initial value of health, and pseudo random values of resistances,
statefactors and weights.

The values of \((H_{s_{\text{max}}})\), \((H_{s_{\text{min}}})\), \((B_{\text{max}})\) and \((B_{\text{min}})\) are specified by the user.

**Axo-Synaptic Branch Life Cycle** This is a scalar processing chromosome. The role of this chromosome is similar to dendrite branch life cycle chromosome. Figure 11 shows the inputs and outputs of axo-synaptic branch life cycle chromosome. It takes health and resistance of the axon branch as input, and produces the corresponding updated values as output. The updated values of resistance are used to decide whether the axon branch should grow, shrink, or stay at its current location. Like the dendrite branches, if the variation in axon resistance is above a certain threshold \((R_{AS})\), it is allowed to migrate to a different neighbour ing location at random. The health of the axon branch decides whether the branch will die, produce offspring, or merely continue with an updated value of health. If the updated health is above certain threshold \((H_{as_{\text{max}}})\) it is allowed to produce offspring and if below certain threshold \((H_{as_{\text{min}}})\) it is removed from the axon. Producing offspring results in a new branch at the same CGPCN grid point connected to the same axon.

The next section will describe the information processing mechanism in the whole network.

### 7 Information Processing in the Network

Information processing in CGPCN starts with the following three steps:

1. Produce a random neural network (the default structure)
2. Produce initial genotypic population, with each consisting of seven chromo-
somes.
3. Specify the number of inputs and output of neural network, and distribute
them at random locations in the network.

In order to produce a random network the following parameters needs to be
specified:

1. The initial number of neurons \((N_i)\)
2. Maximum number of branches \((N_{b_{\text{max}}})\)
3. Maximum number of Dendrites \((N_{d_{\text{max}}})\)
4. Max Neuron state Factor \((N_{s_{f}})\)
5. Max branch state Factor \((B_{s_{f}})\)
6. Mutation rate \((\mu)\)
7. Dimension of 2D space for neurons (Number of rows ($N_{row}$) and columns ($N_{col}$))
8. Neuron and branch life threshold ($H_{min}$), Offspring threshold ($H_{max}$).
9. Neuron and branches health, weight and potentials reduction factors ($\sigma_{H_{db}}$, $\sigma_{H_{as}}$, $\sigma_{W_{db}}$, $\sigma_{W_{as}}$, $\sigma_{P_{db}}$, $\sigma_{P_{s}}$, and $\sigma_{P_{as}}$)

A number of neurons are produced with a random number of dendrites and axon branches. Each dendrite is assigned a number of branches. These neurons are placed at different locations randomly along with their branches in a 2-dimensional CGPCN grid. An initial value of health is assigned to both soma and branches. A random value of weight is assigned to soma and, both dendrite and axon branches. Branches are also assigned with random resistance values. Thresholds are specified for all the operations.

The second part consists of the following tasks:

1. Specify the number of offspring ($\lambda$) (Evolutionary strategy $1 + \lambda$)
2. Number of nodes per chromosome
3. The set of node functions and the number of connection per node.

A random genotype is produced using the above specifications, and evolved to acquire the desired network behaviour.

The third part need to specify is the initial number of inputs and outputs of the system and their corresponding locations in the CGPCN grid.

After producing the initial population the rules for information processing the network is specified. This consists of the following:

1. Input from the environment should be applied to the network at the start of the processing.
2. The network is run for five cycles ($N_{cycles}$) before reading the output.
3. The potential of soma and branches is reduced by ($\sigma_{P_{db}}$, $\sigma_{P_{s}}$ and $\sigma_{P_{as}}$) 10% after every cycle.
4. After five cycles of the network, the health and weights of the soma and branches are reduced by 10%.
5. The soma threshold potential ($\eta_{th}$) is also reduced by double the reduction factor of soma potential after every cycle.
6. The state factor of all the branches and soma are reduced by one unit after every cycle, to allow them to move closer to becoming active.

After specifying the rules for information processing in the network, we can run the network. Network input is applied using the following steps:

1. Find the location of each input.
2. Select the active dendrite branches at that location
3. Bias the input axo-synaptic branch potential. Biasing means duplicating their value in input vectors, equal to number of active dendrite branches, in this case).
4. Apply potentials of active dendrite branches and biased input potential as input to axo-synapse electrical chromosome (CGP Program).
5. The axo-synapse electrical chromosome program updates the values of the dendrite branch potentials (as shown in figure 12)
6. Repeat the same process at each input branch.

After applying the input to CGPCN, run it for a number of cycles (5 in this case). The following steps are performed to run the network for one cycle

1. Select the list of all the active neurons.
2. Start processing each neuron one by one in random sequence

   Each neuron is processed using following steps:
   1. Start processing each dendrite connecting to neuron.
   2. At each dendrite select all the active dendrite branches attached to it, and apply their potential values along with biased soma potential (biasing equal to the number of active dendrite branches) to the CGP dendrite electrical processing chromosome. This produces their updated values as output.
   3. Process each branch potential based on the resistance, weight and health values using equation 4.1 mentioned earlier.
   4. After this process if any of the branch is active, its life cycle is run by applying its resistance, weight and health as input to CGP dendrite branch life cycle chromosome (DBL). This updates these values, and depending on the updated value of resistance the decision to move the branch from the current location is taken. The health decides whether the branch should produce offspring, die or remain the same. Producing offspring means creation of another branch at the same location with an initial health, and random weight and resistance values. If the branch dies it is removed from the dendrite.
5. The same process is repeated for all the dendrites and their corresponding branches. After processing all the dendrites, the average value of potentials of all the dendrites is taken which in turn is the average value of all the active dendrite branches attached to them. This potential and the soma potential are applied as inputs to the CGP soma electrical processing chromosome. This produces the updated value of the soma potential ($\hat{P}$) as output.

6. This updated soma potential is processed using health (H) and weight (W) of soma using equation 4.2 mentioned earlier.

7. After processing the soma potential is compared with the soma threshold potential, if it is higher than the threshold, the soma fires. This means that the soma potential is set to the maximum value. The soma threshold potential and soma statefactor are set to their maximum values (Maximum values are variable depending on user), thus keeping the soma inactive for a number of cycle.

8. If the soma fires, its life cycle is run and also the potential is transferred to other neurons using axo-synaptic branches by running the CGP axo-synaptic electrical processing chromosome (AS).

9. If the soma does not fire, its state factor is adjusted based on the value of the processed potential as described earlier.

10. If the soma life cycle is run, it takes the health and weight of soma as input and produces the updated values as output. If the soma health is below the soma life threshold (one tenth of max in this case) it will die, which means removing neuron from the network along with its branches. If its value is above soma offspring threshold, then it will produce another neuron in the network at the same location with random number of dendrites, branches and other parameters.

Fig. 13. Axo-synaptic potential transfer to the neighbouring dendrite branches, showing soma, two axo-synapse branches, a grid square and a number of dendrite branches attached to other neurons and their corresponding potentials, and a weight processing chromosome. The dark circle represents the soma electrical processing chromosome (S), light cyan circles represents weight processing chromosomes (WP), and blue circle axo-synapse (AS). Red bars above circles and lines shows the potential values of axo-synapse and dendrite branches. The dotted green (blue) lines represents dendrite (axo-synapse) branches of the other neurons in the network.
11. If the soma fires the signal needs to be transferred to other neurons. This is done as follows: the axo-synaptic cgp electrical processing chromosome is run, in the active axon branches (as show in figure 13). Select all the active dendrite branches in the vicinity of each active axon branch and take their potential values, and apply them together with a biased soma potential (equal to number of active branches) as inputs to the axo-synaptic CGP electrical processing chromosome. The chromosome produces the updated potentials of all the dendrite branches, along with axo-synaptic potentials. After this process the axo-synaptic potential is processed using equation 4.3 as mentioned earlier.

![Fig. 14. Weight processing of the neighbouring dendrite branches, showing soma, two axo-synapse branches, a grid square and a number of dendrite branches attached to other neurons and their corresponding weights, and weight processing chromosome highlighted in the grid square. The dark circle represents the soma electrical processing chromosome (S), light cyan circles represents weight processing chromosome (WP), and blue circle axo-synapse (AS). W_i shows the weights of axo-synapse and dendrite branches.](image)

After getting the processed potential the weight processing CGP chromosome is run, this takes the weights of the active dendrite branches in the vicinity of the axon branch and its axo-synaptic weight as input and produces the updated values as output as shown in figure 14. The axo-synaptic potential is assigned to the dendrite branch whose weight is the maximum after weight processing as shown in figure 15.

12. Also after the axo-synaptic electrical processing, if the potential of the axo-synaptic branch is raised above a certain level, it is kept active and its life cycle is run.

13. The life cycle of axo-synapse takes health and resistance of the axon as input and produces their updated values. The change in resistance of the branch is compared with a threshold value to decide whether the branch should move or stay at the same location. When the health of the branch if is above the offspring threshold another branch is produced at the same location, with an initial health and random weight and resistance. If the health falls below the life threshold it dies and is removed from the axon.
Intelligent agents capable of developing memory of their environment

Fig. 15. Transfer of potential to highest weight after weight processing. The diagram shows a soma, two axo-synapse branches, a grid square and a number of dendrite branches attached to other neurons and their corresponding weights, and a weight processing chromosome. The highest weighted branch is highlighted where the axo-synapse potential is transferred in the grid square.

14. The same process is repeated in all the axon branches

After running the network for five cycles, the output is read from the output branches. The output branches are affected by the network processes, through their updated potential values.

After completing the task the following steps are performed:

1. The network fitness is assessed.
2. The genotype with highest fitness is selected.
3. Using mutation new offspring chromosomes are produced.

8 Wumpus World

Wumpus World, is a variant of Gregory Yobs “Hunt the Wumpus” game [Yob, 1975]. It is an agent-based learning problem used as a testbed for various learning techniques in Artificial Intelligence [Russell and Norvig, 1995]. Wumpus World was originally presented by Michael Genesereth [Russell and Norvig, 1995]. It consists of a two dimensional grid containing a number of pits, a wumpus (monster) and an agent [Russell and Norvig, 1995] as shown in figure 16.

An agent always starts from a unique square (home) in a corner of the grid. The agent’s task is to avoid the wumpus, and pits and find gold, return it to home. The agent can perceive a breeze in squares adjacent to the pits, a stench in the squares adjacent to the wumpus, and a glitter on the gold square. The agent can be armed with one or more arrows. So it can shoot the wumpus by facing it. In most environments there is a way for the agent to safely retrieve the gold. In some environments, the agent must choose between going home empty-handed, taking a chance that could lead to death, or finding the gold. Wumpus World can be presented in different environments ([Yob, 1975] used an environment that was a flattened dodecahedron), but the most common is the rectangular
Intelligent agents that develop a memory of an environment.

Fig. 16. A two dimensional grid, showing Wumpus World environment, having one wumpus, one agent on the bottom left corner square, three pits and gold. Spector and Luke investigated the use of genetic programming to solve the Wumpus World problem [Spector, 1996, Spector and Luke, 1996]. However, they just evolved fixed programs that controlled the agent, so there was no learning and the agent did not have a lifetime.

8.1 Experimental setup

In our experiments we have slightly adapted the rules of the Wumpus World. Namely, the agent encountering pits or wumpus is only weakened (thus reducing its life), it is not killed and the agent does not have any arrows to shoot the wumpus. Also a signal called ‘glitter’ can be detected on squares neighbouring the gold (north, south, east, west). These changes are introduced to facilitate the capacity of the agent to learn. The CGPCN learns everything from scratch and builds its own memory (including the meaning of signals, pits and the wumpus).

It is important to appreciate how difficult this problem is. The agents starts with a few neurons with random connections. Evolution must find a series of programs that build a computational network that is stable (doesn’t lose all neurons or branches etc.). Secondly, it must find a way of processing infrequent environmental signals. Thirdly, it must navigate in this environment using some form of memory. Fourthly, it must confer goal-driven behaviour on the agent. This makes the wumpus world problem a challenging problem.

The Wumpus World we used is a two dimensional grid (10x10), having ten pits, one wumpus and the gold. The location of the wumpus, gold and pits is chosen randomly. In the square containing the wumpus, gold, pits, and in a directly (not diagonally) adjacent square the agent can perceive stench, glitter and breeze, respectively. The input to the CGPCN is a potential of zero, if there is nothing on the square, a potential of 60 if encounters a pit, 120 if caught...
by wumpus and 200 if arrives at the square containing the gold. The system is arranged in such a way that the agent will receive input signals of different magnitudes depending on the direction that the agent perceives the signal. The agent’s CGPCN can perceive the direction of a breeze through the magnitude of the breeze potential. We chose the following values to represent these directions: north of the pit is 40, east 50, south 70, and 80 for west. Similarly the stench signal has a direction which is represented by different magnitudes of potentials as follows: for north 100, for east 110, for south 130 and finally 140 for west of wumpus. Also it receives 180, 190, 210 and 220 from north, east, south and west of gold on for glitter. An agent detects that it is on the home square via a special input signal (255 is the maximum value of potential). In all the other locations which are safe the input potential is zero. The agent can always perceive only one signal on a grid square, even if there are more than one. The priority of the signals is in the following order: wumpus, gold, pit, stench, glitter, breeze. The agent is assigned a quantity called energy, which has an initial value of 200 units. If an agent is caught by the wumpus its energy level is reduced by 60%. If it encounters a pit its energy level is reduced by 10 units. If it gets the gold its energy level is increased by 50 units. Finally, on arriving home the agents cease to exist. For each single move the agent’s energy level is reduced by 1 unit, so if the agent just oscillates in the environment and does not move around and acquire energy through solving tasks it will run out of energy and die.

The fitness of an agent is calculated according to its ability to complete learning tasks. It is accumulated over the period that the agents energy level is greater than zero (or before it returns home). The fitness value, which is used in the evolutionary scheme, is accumulated while the agent’s energy is greater than zero as follows:

- For each move, the fitness is increased by one. This is done, to encourage the agents to have ‘brain’ that remains active and does not die.
- If the agent returns home without the gold, its fitness is increased by 200.
- If the agent obtains the gold, its fitness is increased by 1000.
- If the agent returns home with the gold, its fitness is increased by 2000.

When the experiment starts, the agent takes its input from the Wumpus World grid square. This input is applied to the computational network of the agent and is processed by input axo-synapses. The network is then run for five cycles (one step). During this process it updates the potentials of the output dendrite branches which act as the output of the CGPCN. After the step is complete the updated potentials of all output dendrite branches are noted and averaged. The value of this average potential decides the direction of movement for the agent. If there is more than one direction the potential is divided into as many ranges as possible movements. For instance if two possible directions of movement exist, then it will take one direction if the potential is less than 128 and the other if greater. The same process is then repeated for the next Wumpus World grid square. The agent is terminated if any of the following conditions occur: its energy level becomes zero; all its neurons die; all the dendrite or axon branches die; the agent returns home.
Fig. 17. Structural changes in an agent’s CGPCN network at different stages in Wumpus World shown against number of completed steps. The initial network has 5 neurons and 21 neurons after completing 80 steps. Black squares are somas, red thick lines are dendrites, yellowish green lines are axons, green lines are dendrite branches, and blue lines show axon branches.

Five randomly generated genotypes are produced and the corresponding agent behaviour is assessed to obtain its fitness. The best agent genotype is selected as the parent for a new population. The CGPCN is arranged in the following manner for this experiment. The neurons are confined to 3x4 CGPCN grid. Inputs and outputs to the network are located at five different random squares. The initial number of neurons is 5. The maximum number of dendrites is 5. The maximum branch statefactor is 7. The maximum soma statefactor is 3. The mutation rate is 5%. The maximum number of nodes per chromosome is 100.

8.2 Results and Analysis

We carried out twenty independent evolutionary runs (using the same Wumpus World) and found the average numbers of generations that the agent took to return home without gold was 4, to find the gold was 13 and to return home with gold was 300.

While solving the Wumpus World the CGPCN changes substantially in its structure. Figure 17 shows the variation in neural structures and growth of the CGPCN at different stages.

To test whether agent behaviour was general we took the programs for the best evolved agent on the original Wumpus World and tested its performance on other wumpus worlds generated at random. We found that in some of the cases the agent was able to get the gold and bring it to home. In others the agent gets the gold but is unable to find its way to home. Sometimes it cannot find the gold and returns home empty handed. In others, the agent cannot find the gold or get back home. Interestingly, we found that the agent always first
Intelligent agents capable of developing memory of their environment looks for the gold at the place where the gold was when it was evolved. This is very interesting as it shows that the evolved genetic codes are able to build a computational network that holds information about where to find the gold from an initial small random network holding no information. This is reminiscent of instinctive behaviour.

We also examined the learning behaviour of a number of highly evolved agents when they were allowed to live after returning home (this has not been allowed in the evaluation of their fitness). Figure 18 shows the original path that the agent took to the gold, when evolved (on left) and the three other paths to the gold on subsequent journeys. The agent takes 135 steps to find the gold in the first of these journeys. On the second journey (Fig 16b) the agent took an almost straight path towards the gold and encounters a pit once. On the third journey (Fig 16c) the agent takes almost the same path as the second journey, but with more energy wasting oscillations. It encounters two pits, but finally gets the gold. In the fourth and final journey the agent follows a similar path to that of third, but with even more oscillations and when it reaches a corner it gets stuck until its energy level decays to zero.

It is worth noting that an agent that moves in a straight line (even at edges) is highly non-random. A random move at an edge would have 33% probability. Thus a straight path of 8-steps (as in figure 16b) returning home would have a probability of 0.0152% which means that straight paths are very unlikely if the agent is merely choosing directions at random.

9 Conclusion

We have presented a brain-inspired computational system tested in a classic AI learning problem. We found that agents could acquire an understanding of the meaning of environmental signals and act intelligently based on them. In addition, the agents exhibited a form of memory that arose as a consequence of the
interaction of their internal developmental processes and external environmental signals. We also tested the system without development (by ignoring the life cycle chromosome), and found that it took much longer to evolve networks with the same performance as the original system. This highlights the importance of development. In future work, we plan to evaluate this approach in richer and more complex environments.

References


