Amygdala – Genetic Algorithm

(Genetic Algorithm to evolve adapded Amygdala neural networks)

Rüdiger Koch rkoch@rkoch.org

Jan. 28, 2002 (version 0.1)

Abstract: A Genetic Algorithm allows for semi automatic creation of large Amygdala spiking neural networks which are adapted to a given problem. Design issues and implementation of the GA infrastucture are discussed.

1.Intro

A Spiking Neural Network (SNN) is lacking the large number of learning algorithms, particularly supervised learning, that are available for traditional artificial neural networks. The only learning algorithm presently implemented in Amygdala is Hebbian learning. This seems to be a disadvantage at first. On the other hand the traditional supervised methods are biologically implausible. What's more, they don't scale well, making these algorithms unsuitable for large networks.

How does learning work with us? Learning involves certain abstract entities we all believe we know what they are but have trouble to define. These are consciousness, pain and pleasure. If something new approaches us we are directing our consciousness towards it. Then we evaluate it in terms of good and bad. Learning takes place. We are clearly getting feedback during this process. If we assume that all learning in a biological brain is local, how can such feedback influence learning, as it undouptedly does?

The answer might lie in the microscopic and macroscopic structure of the brain. The assumtion this paper builds upon is that in the space of all possible topologies there are small regions that allow feedback to work together with local learning in such a way that the SNN adapts to the problem it is exposed. This is opposed to the view that the brain of a worm or an insect's synapses are determined by it's genes. Such a view cannot hold: The genome of a fruit fly is not much more than 100MByte of data. This is not enough to encode all (or even

a significant number) of all synapses of the fly's 200,000 neurons. The brain of a fly is similar in size of what we are trying to achieve: SNN topologies of the flybrain's size that are suited to solve the problems they are evolved for in a similar way as the fly's development solved the problem to survive in the fly's environment.

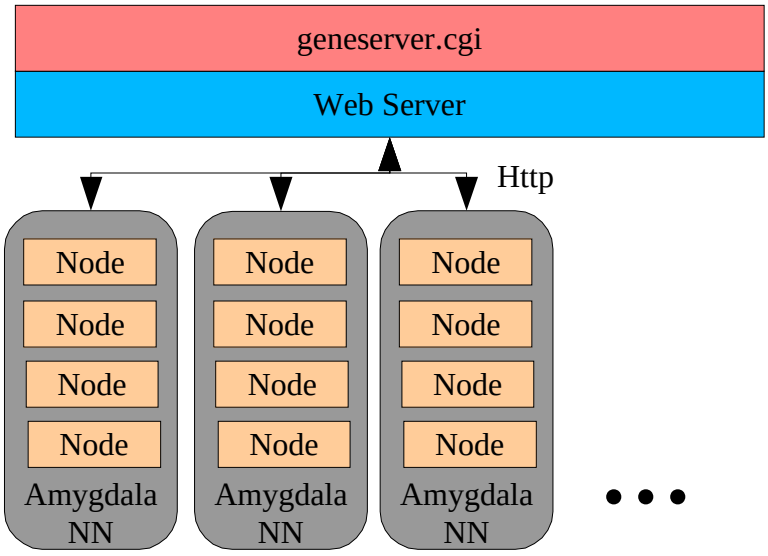
Our task is to find a fitting topology. A suitable algorithm to search the space of all possible topologies is the Genetic Algorithm (GA). A GA essentially mimics nature. Individuals are generated from genes. The individuals are tested for fitness and selected for reproduction. Reproduction is done by merging and mutating genes. The GA presented here differs substantially from general purpose packages such as GENESIS in that it is rather simple. On the other hand it is designed to utilize the parallelism of networks of workstations.

The final goal of this development is to allow for an automation of programming. The user's task would be limited to creating an environment that represents the problem. Amygdala then finds the solution by self organization.

2. Overall Architecture

Finding a suitable SNN topology that fits to our problem or set of problems on hand is an extremely computing intensive task. Therefore our architecture must be able to solve this task in a distributed environment. The most computing intensive part is the evaluation of an individual since this means no less than running an Amygdala SNN long enough to make a decision of how good this individual is. The management of the genes is a much less daunting task.

We separate the task into a client/server achitecture. A geneserver manages the genes which clients download and evaluate. After evaluation, the clients report the result back to the server.



The clients communicate with the server through http. The geneserver is actually only a CGI program, started by a Web server on request. This architecture allows for an almost unlimited number of clients. The clients may use the Amygdala MP extensions to allow for larger networks.

The Genome format is constant for the entire simulation. It must be defined at the beginning.

3Client / Server

3.1The Gene Server

As already mentioned, the gene server is not a server by it's own right but a CGI program. As such it requires a Webserver for operation. All operations are file based.

When a client requests a gene the server first looks if a gene is available. If not, there are two choices. Either sufficient clients already reported their results so there are enough genes available to create a new generation or a random gene of those that are currently evaluated is duplicated. The gene is then written to stdout in normal CGI fashion.

3.2The Client

The client has 2 layers. The first layer is implemented by the `Genome` class. It's task is to do the communication with the server and scan the genome string. It leaves it to a particular `GenomeHandler` class to create an Amygdala SNN (phenotype). There can be many models about how to do this. Provided with the Amygdala library is the Cherrymoya

3.2.1The Genome class

The `Genome` class communicates with the server, using the protocol described below. It downloads a genome, chops it into chromosomes and genes and hands the genes to the phenotype implementation that has been registered with it.

3.2.2Implementing a phenotype

The `GenomeHandler` class defines the transformation of the genome to a phenotype (an Amygdala SNN). There are a wide range of options such as doing the transformation probabilistic or deterministic. The method of deriving a topology from the gene can vary. Very straight methods can be imagined or methods that simulate growth...

Plugging in a new phenotype is possible. The only requirement is that class implementing it derives from class `GenomeHandler` and implements the abstract methods defined in there.

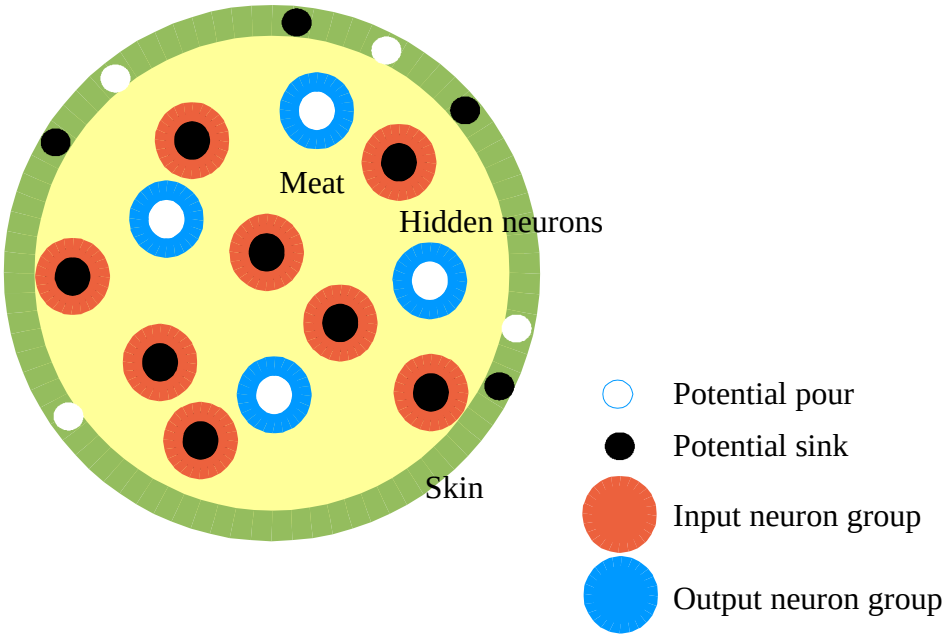
3.2.3 The Cherrymoya phenotype

The Cherrymoya model uses a deterministic method of transforming a gene into a SNN. It uses a fixed amount of neurons in a sphere of radius r. Within this sphere, input and output neurons are placed as determined by their genes. The result looks like a Cherrymoya fruit. Input and Output neurons have potential pours and sinks associated to them which are defined by:

$$\vec{E} = \frac{e}{r^3}\vec{r}$$

The strength E of the field determines the length of the axon. The direction of the flow determines the direction of the axon. So the interior of the sphere can be imagined as a flow field with the axons forced along the streamlines.

The transformation is motivated by electodynamical / fluid mechanical flow fields which are solutions of the Laplace equation. There is no deeper physical reason for that, however, other than trying to create a complex structure from a simple genome.



The surface of the sphere is covered by a one neuron thick skin of hidden neurons. Purpose of this layer is to reflect spikes from within the sphere back into to sphere. Neurons in the skin have a devided axon. One part points to the center of the sphere, the other part runs around the sphere according to

$$\vec{E} = \frac{e}{r^2}\vec{r}$$

E again determines the length of the axon. The distance from the singularity is r

3.3The Protocol

Some protocol information is added. The request can be either with or without

arguments in the GET method. No arguments mean that a new genome is requested which is then delivered. The client can report the result of an evaluation with a request string in the form:

```
GET /path/to/cgi−bin/geneserver?genomeID&score
```

where genomeID and score must be decimal numbers. The genomeID must be the same as given when the Genome was requested. The score must be smaller than 2^{32}. Larger score means better genome. An example string is:

```
GET /cgi−bin/geneserver?543&2834
```

When a genome is requested, additional header information is given:

```
Chromosome−format: {n1,m1}{n2,4m2}{n3,m3}....
Genome−Id: N
```

N is the Genome−Id as explained above. The nx are the number of genes in a chromosome, the mx is the size (in bytes) of a gene of this chromosome.