Neuroevolution Through Erlang

Erlang Factory
San Francisco – March 2012

Gene I. Sher

The Figures/Images in this slide presentation are taken from:
”**Handbook of Neuroevolution Through Erlang”** 2012, Springer.

Talk Outline

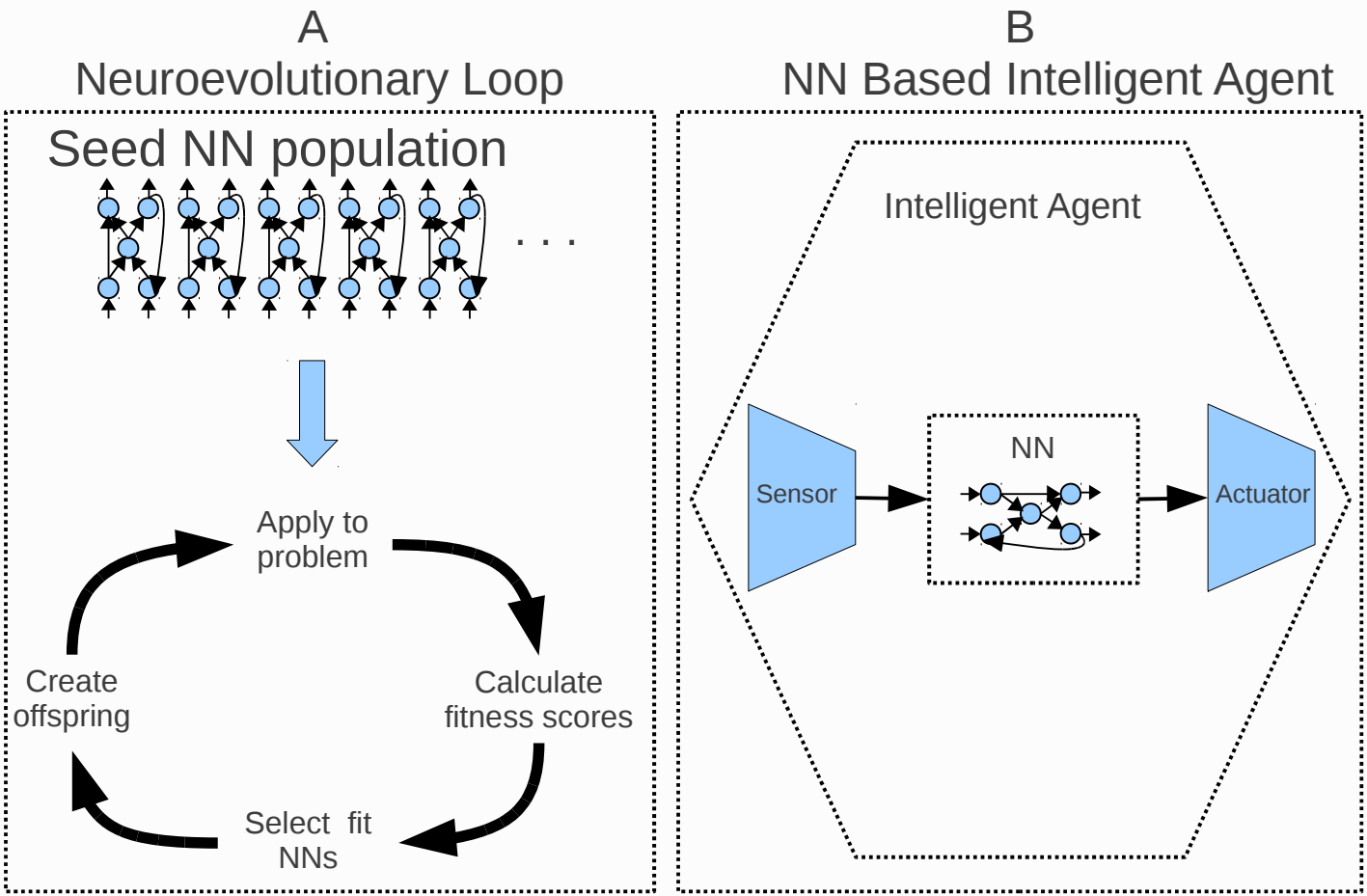
- The General Idea
- Applications & Motivations
- Background
  - Neural Networks
  - Evolutionary Algorithms
  - Neuroevolution
- Erlang: The Quentisential Neural Network Programming Language
  - The necessary elements
  - The architecture
  - The mapping
- A Case Study
  - The Genotype
  - The Phenotype
    - Exoself
  - The Algorithm
  - The Scape
    - Infomorph
  - Substrate Encoding
- Demonstrations of Applications
  - Artificial Life
  - Currency Trading
- Ongoing and Future Projects
  - Cyberwarfare
  - CPU Evolution
  - UCAV Neurocontrollers
  - NNRR
- My Book: ”Neuroevolution Through Erlang”, to be released by Springer towards the end of 2012.

The General Idea

A bootstrap for the next section

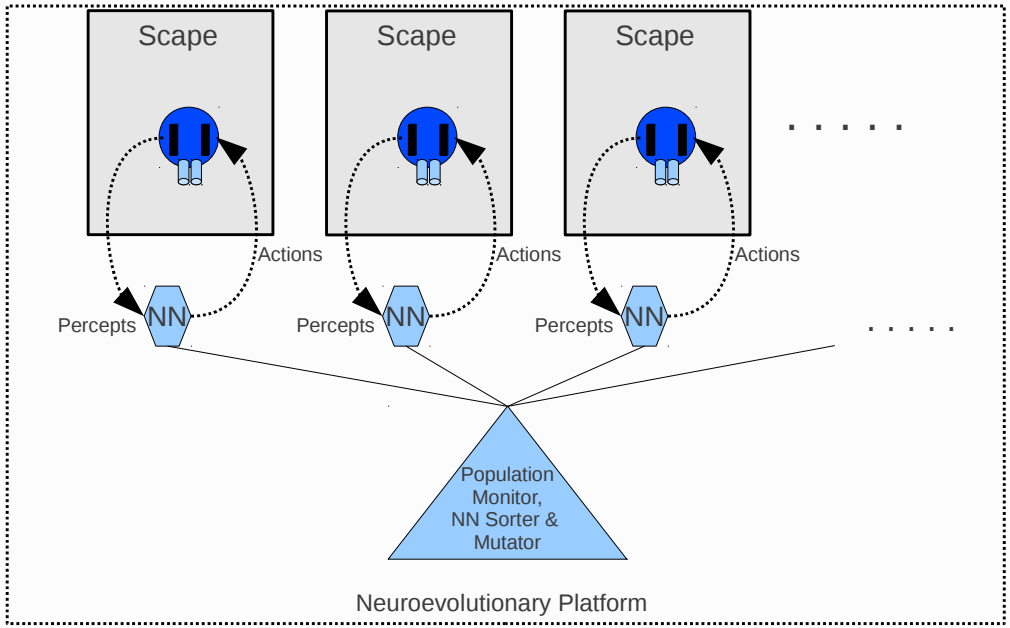
- Neural Networks are just groups of interconnected nodes/artificial neurons/neurodes
- An artificial neuron accepts an input signal, processes it, and produces an output signal.
- Multiple neurons can be connected to gether to produce complex processing systems.
- Such a system is called a neural network
- We can mutate, by modifying either its topology or other parameters
- Then see if it performs better or worse and go from there
- A NN with sensors and actuators is also called an intelligent agent

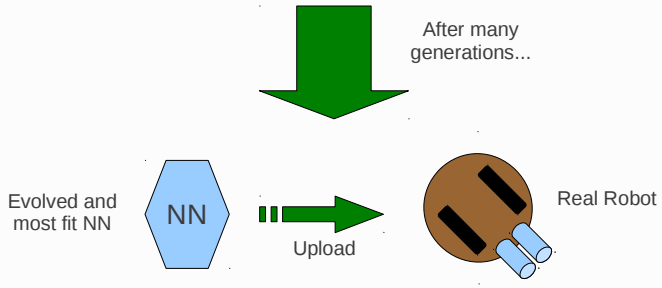
Evolution of intelligent agents



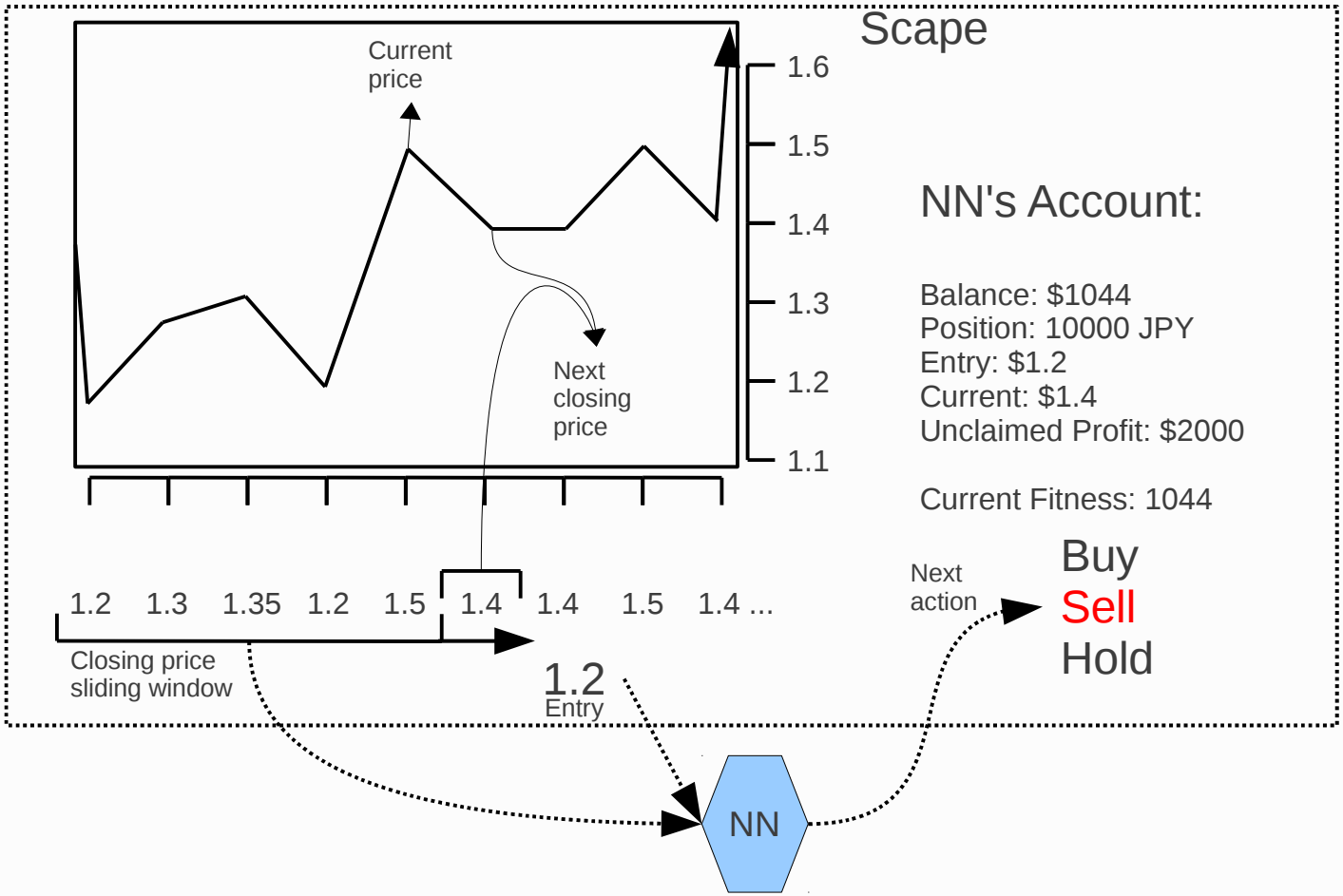
Applications & Motivations
The *why* behind the subject computational intelligence.

Robotics


Scape
Scape
Scape
Actions
Actions
Actions
Percepts
NN
Percepts
NN
Percepts
NN
Population Monitor, NN Sorter & Mutator
Neuroevolutionary Platform


After many generations...
Evolved and most fit NN
NN
Upload
Real Robot

Financial Analysis



Artificial Life

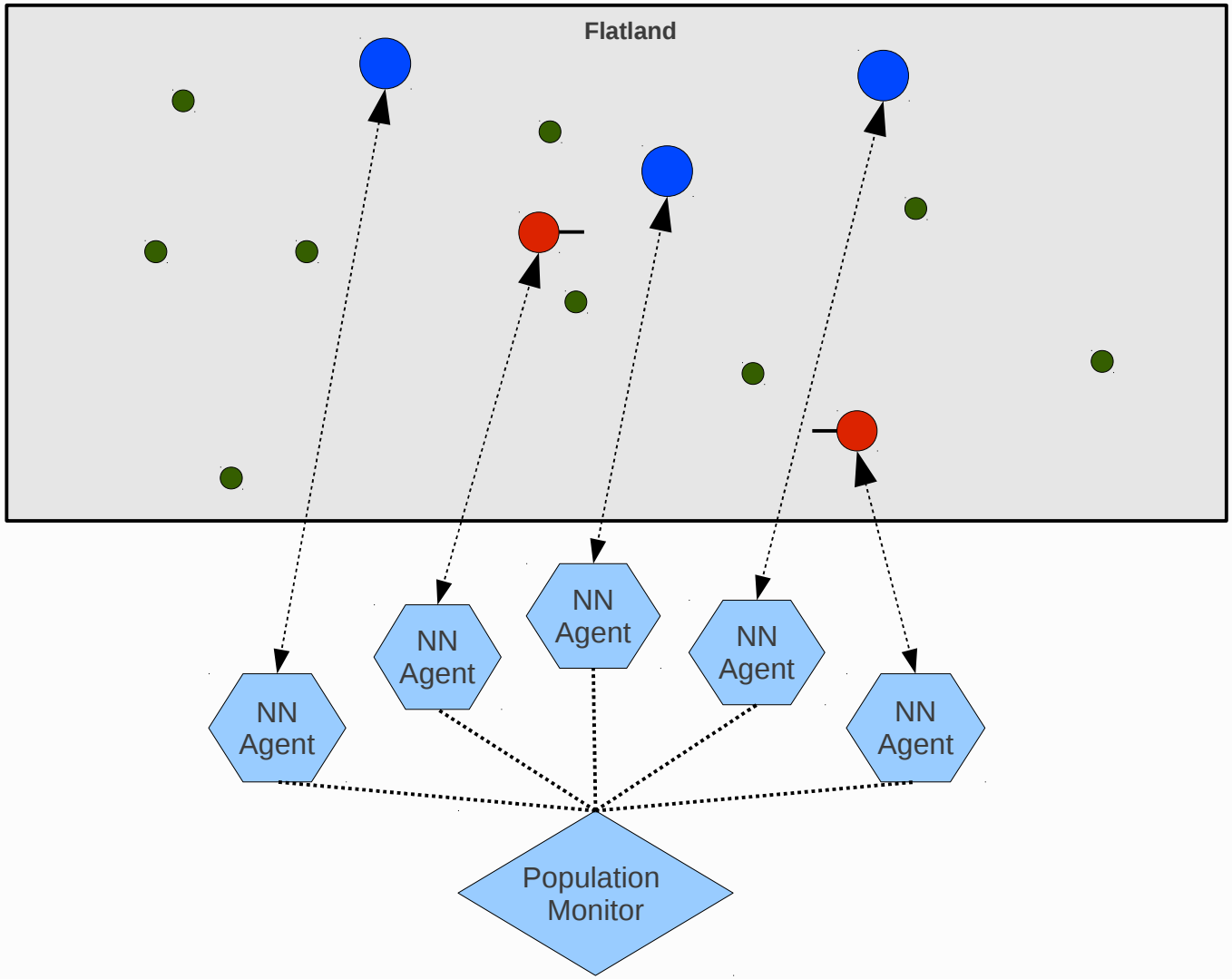
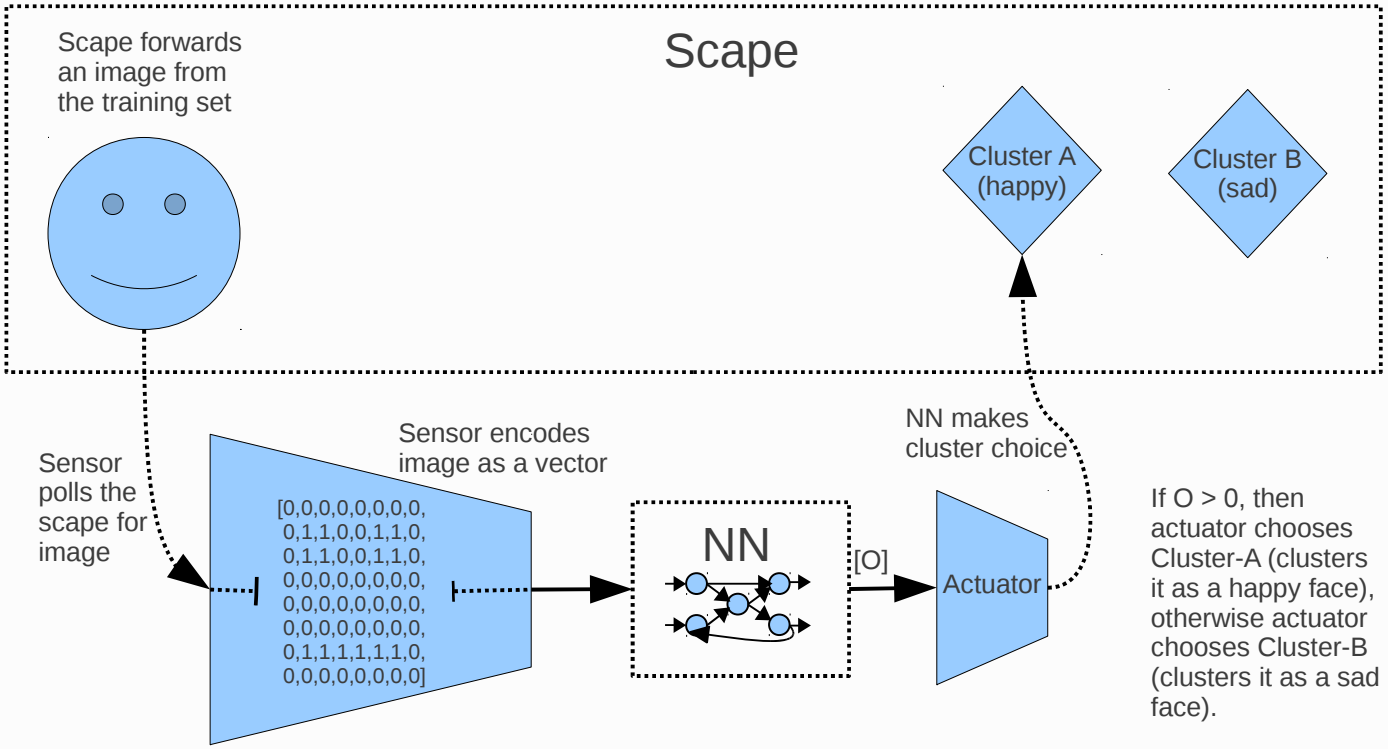
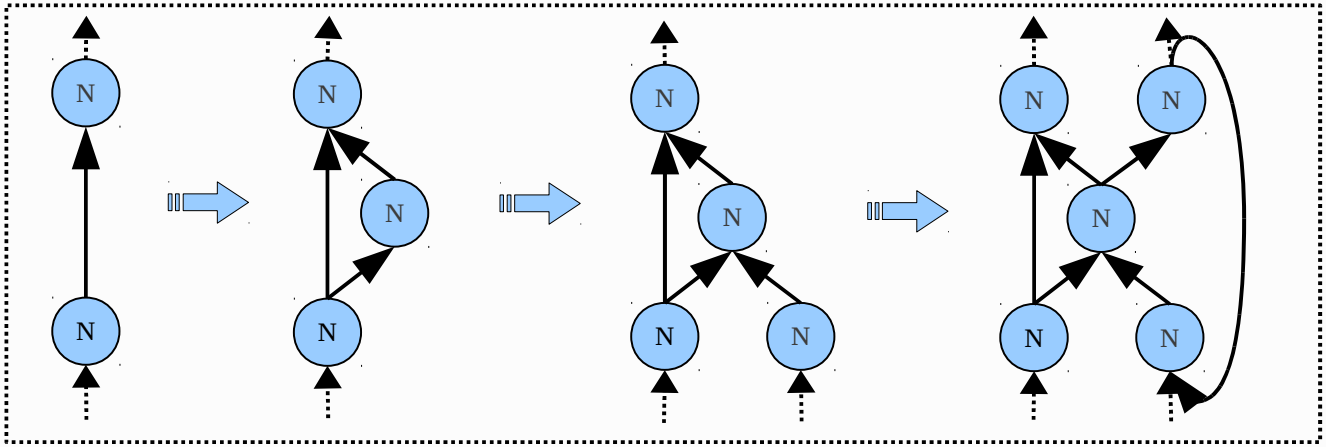


Image Analysis

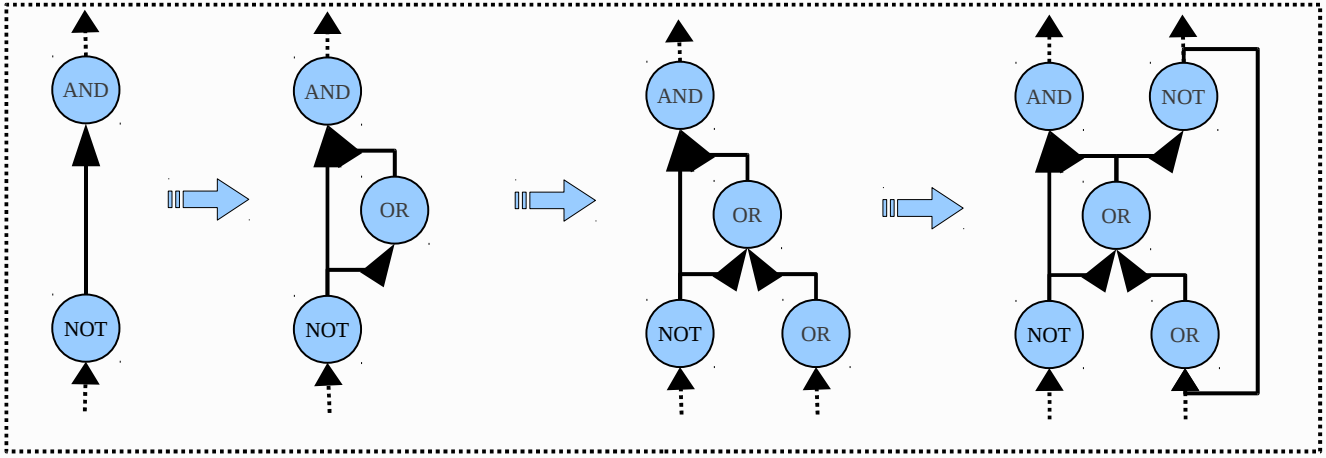


Evolving Circuits

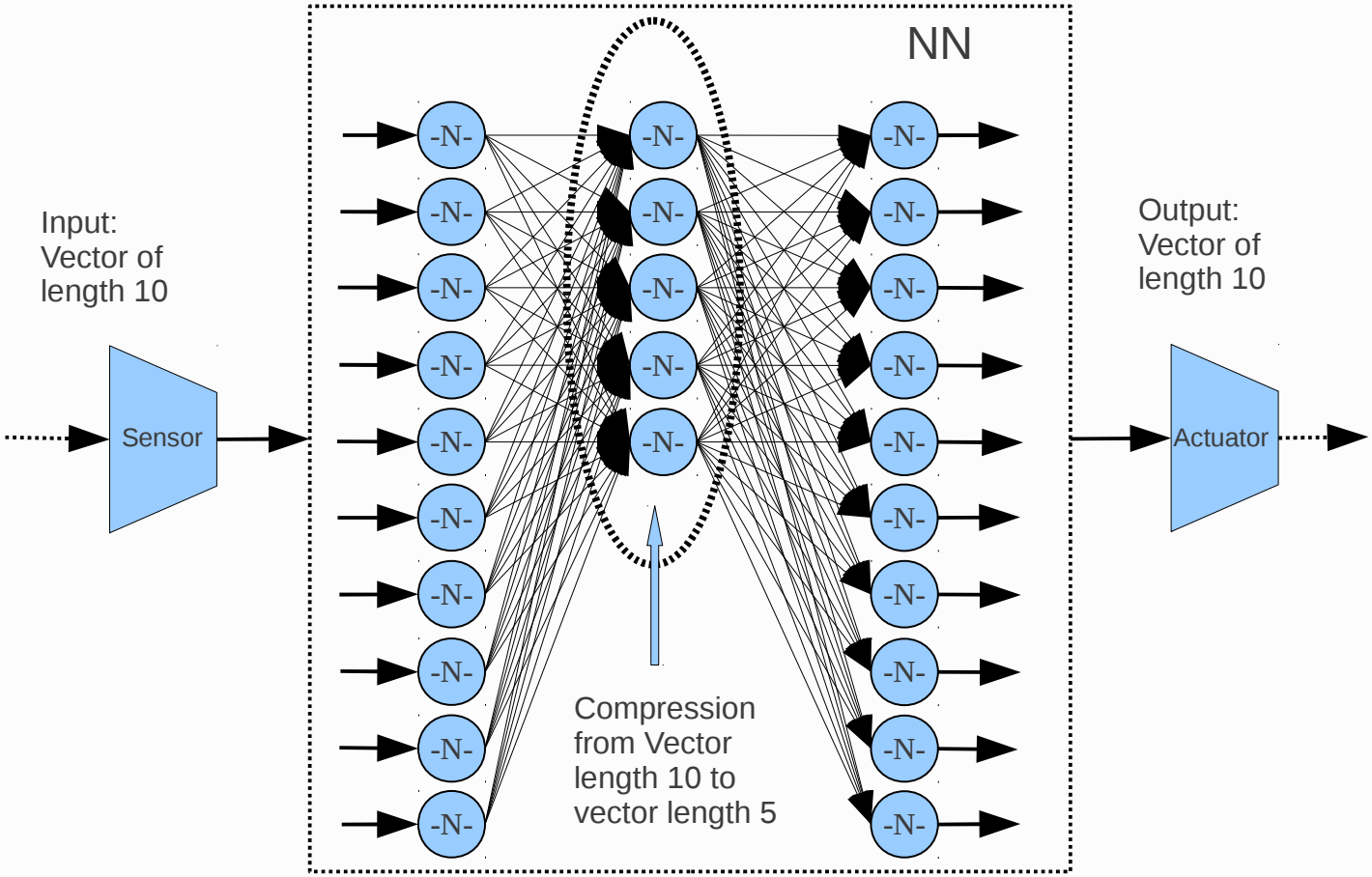
Evolving Neural Network Topologies



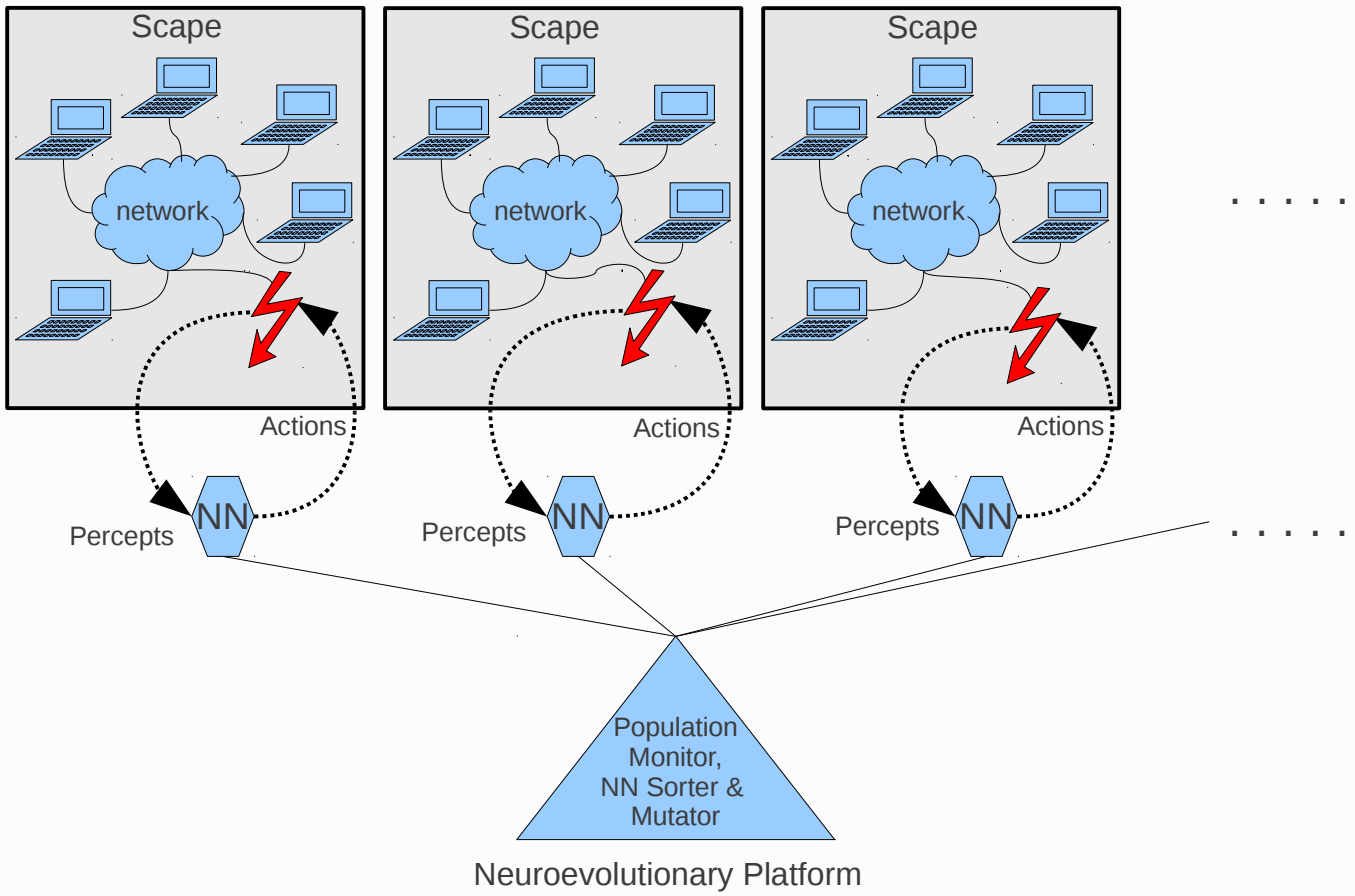
Evolving Digital Circuit Topologies



Data Compression



Cyber Warfare



Endgame Singularity

- The Long Standing Goals of Computational Intelligence
- Brain is just an organic-substrate based NN
  - Blue Brain Project - http://bluebrain.epfl.ch/
  - MoNETA (MOdular Neural Exploring Traveling Agent) project (from Boston University's Neuromorphics Lab)
  - OpenCog Project
  - Hugo de Garis, Shuo Chen, Ben Goertzel, Ruiting Lian: A world survey of artificial brain projects, Part I: Large-scale brain simulations. Neurocomputing (IJON) 74(1-3):3-29 (2010)
  - Ben Goertzel, Ruiting Lian, Itamar Arel, Hugo de Garis, Shuo Chen: A world survey of artificial brain projects, Part II: Biologically inspired cognitive architectures. Neurocomputing (IJON) 74(1-3):30-49 (2010)
  - DARPA's SyNAPSE project
- Computer hardware is advancing steadily
- A programming language is needed with the right features
- Erlang: As the NN Programming Language

Background

- I will make a case for the use of Erlang within the field of Computational Intelligence research and development.
- Some background information is needed.
- We will briefly cover:
  - Biological neural networks.
  - Artificial neural networks.
  - Evolutionary computation.
  - Neuroevolution.

Neural Networks

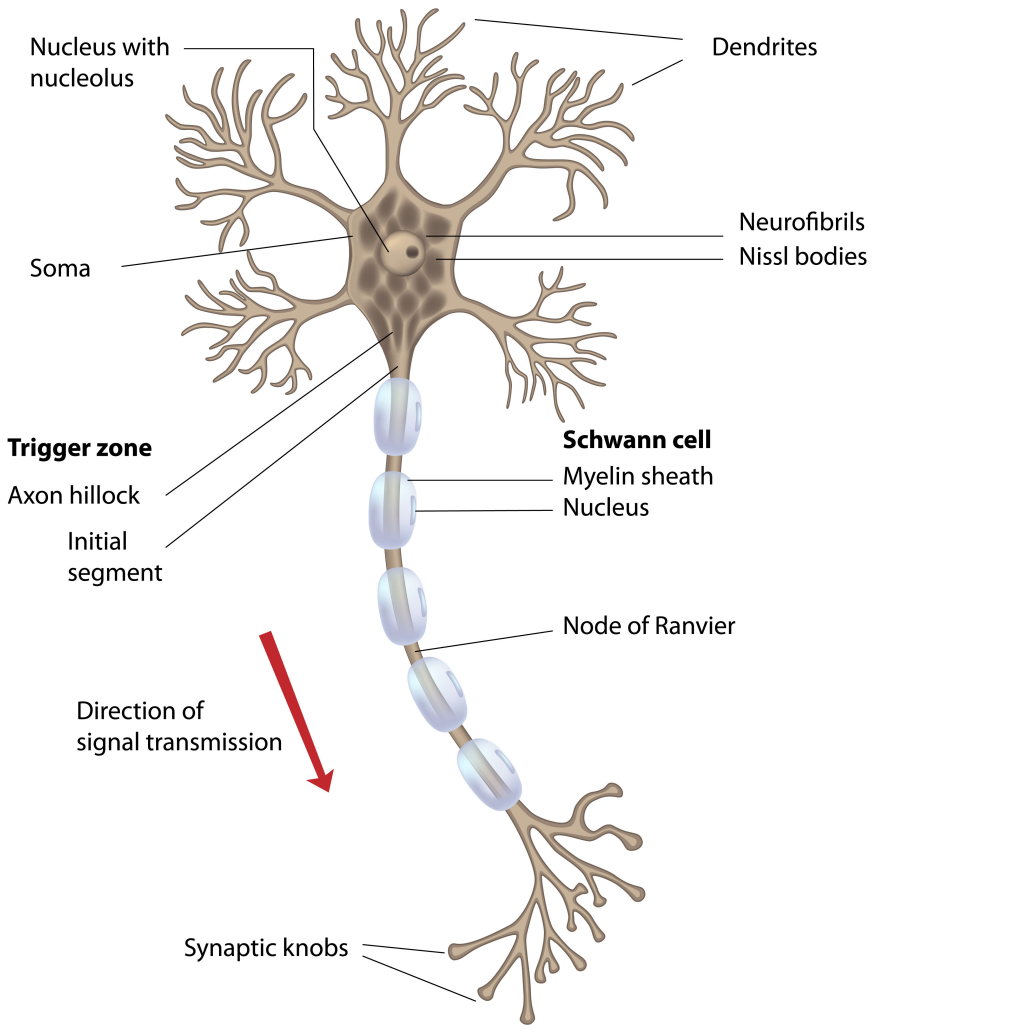
- Vast networks of signal processors
- Concurrent processing
- Power in numbers
- Robust and fault tollerent
- Recovery from damage

Biological Neural Network

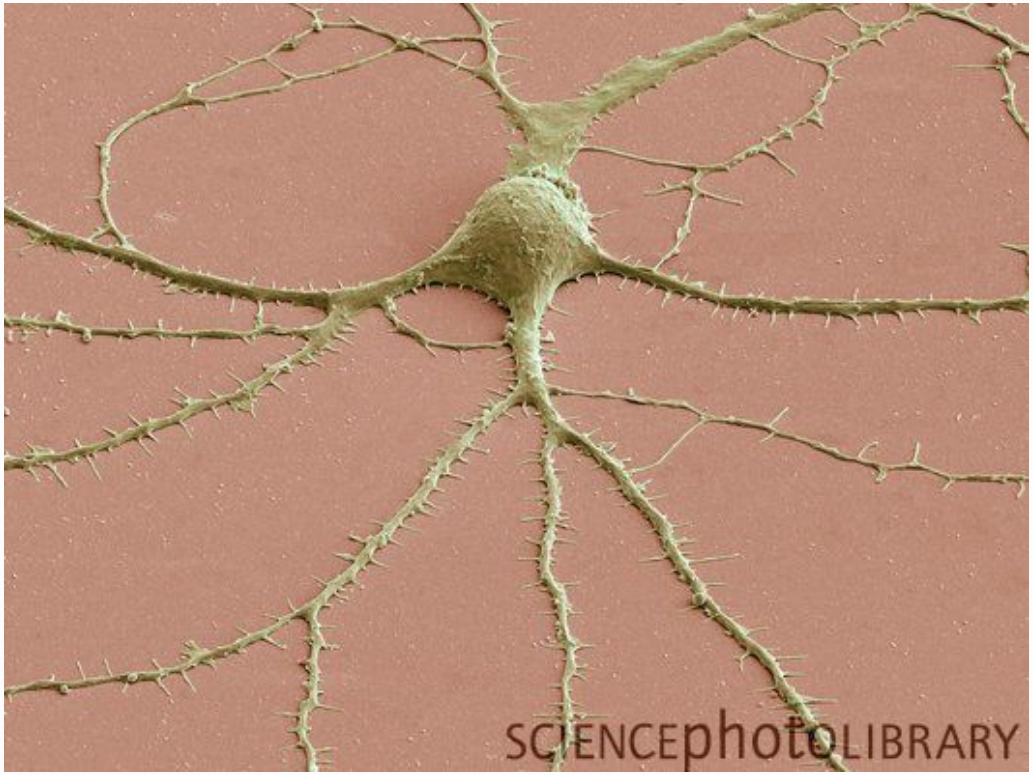
Biological neuron

- An biological processing node
- Signal integration
- Spatiotemporal signal processing
- Frequency encoding
- Biological limitations

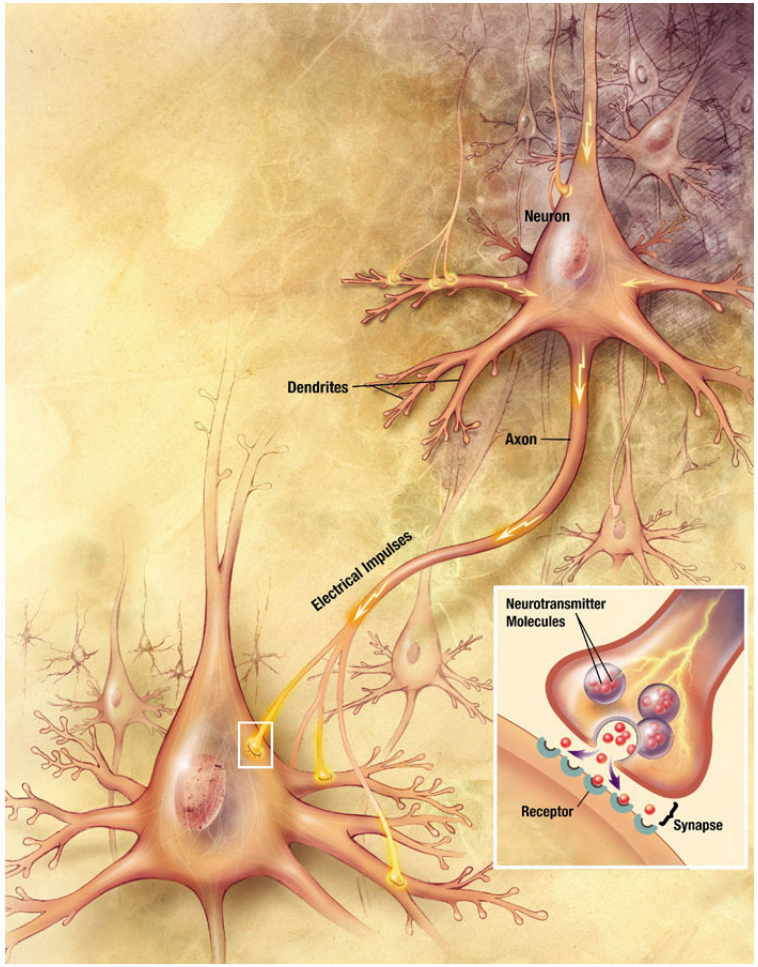
A multipolar neuron (Ex. spinal motor neuron)



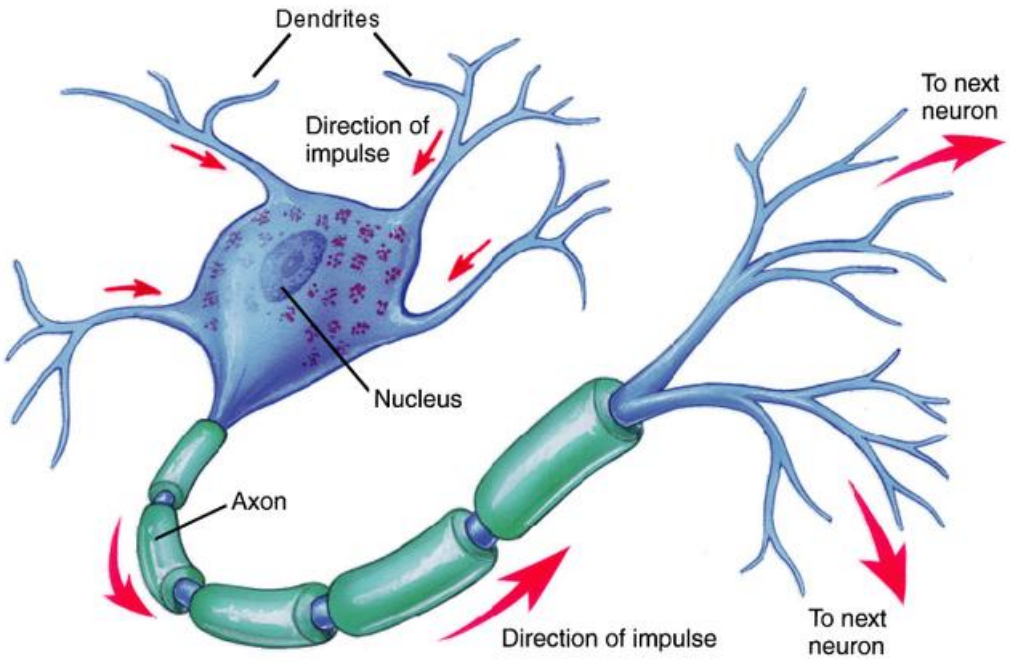
Actual Biological Neuron

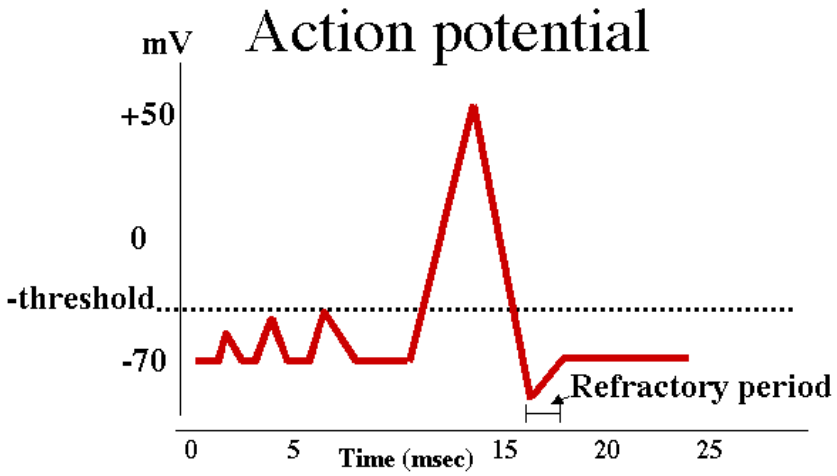


Signal Integration



Frequency Encoding





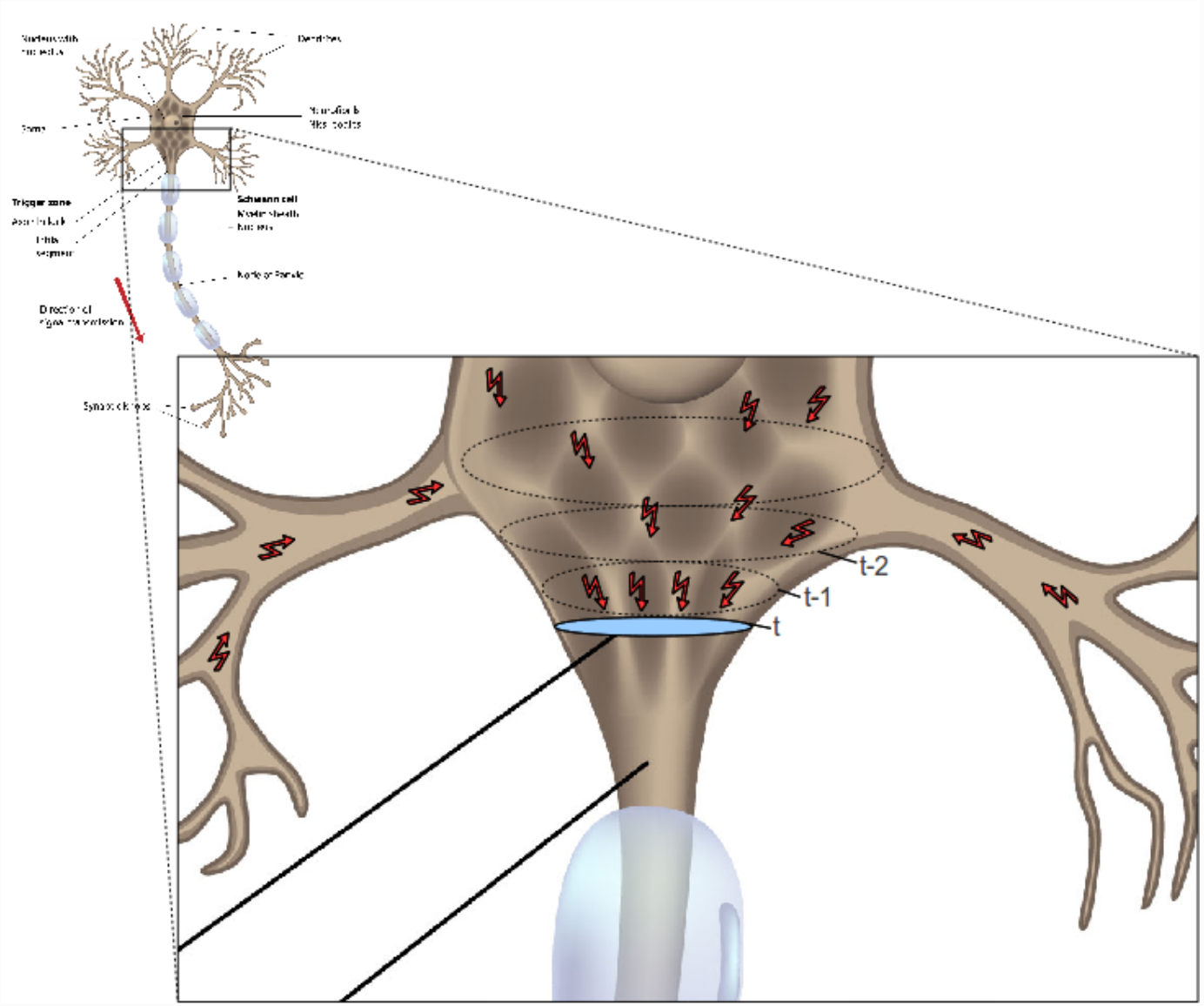
Threshold- to generate an action potential an axon requires a stimulus of a certain minimum strength

All or none - each action potential has the same amplitude independently from the strength of the stimulus

Refractory period- a second action potential cannot occur during this period

Whether the dendrites experience excitatory or inhibitory signals, depends not only on the actual signal sent by the presynaptic neuron, but also on the dendrites, their chemistry, receptors...

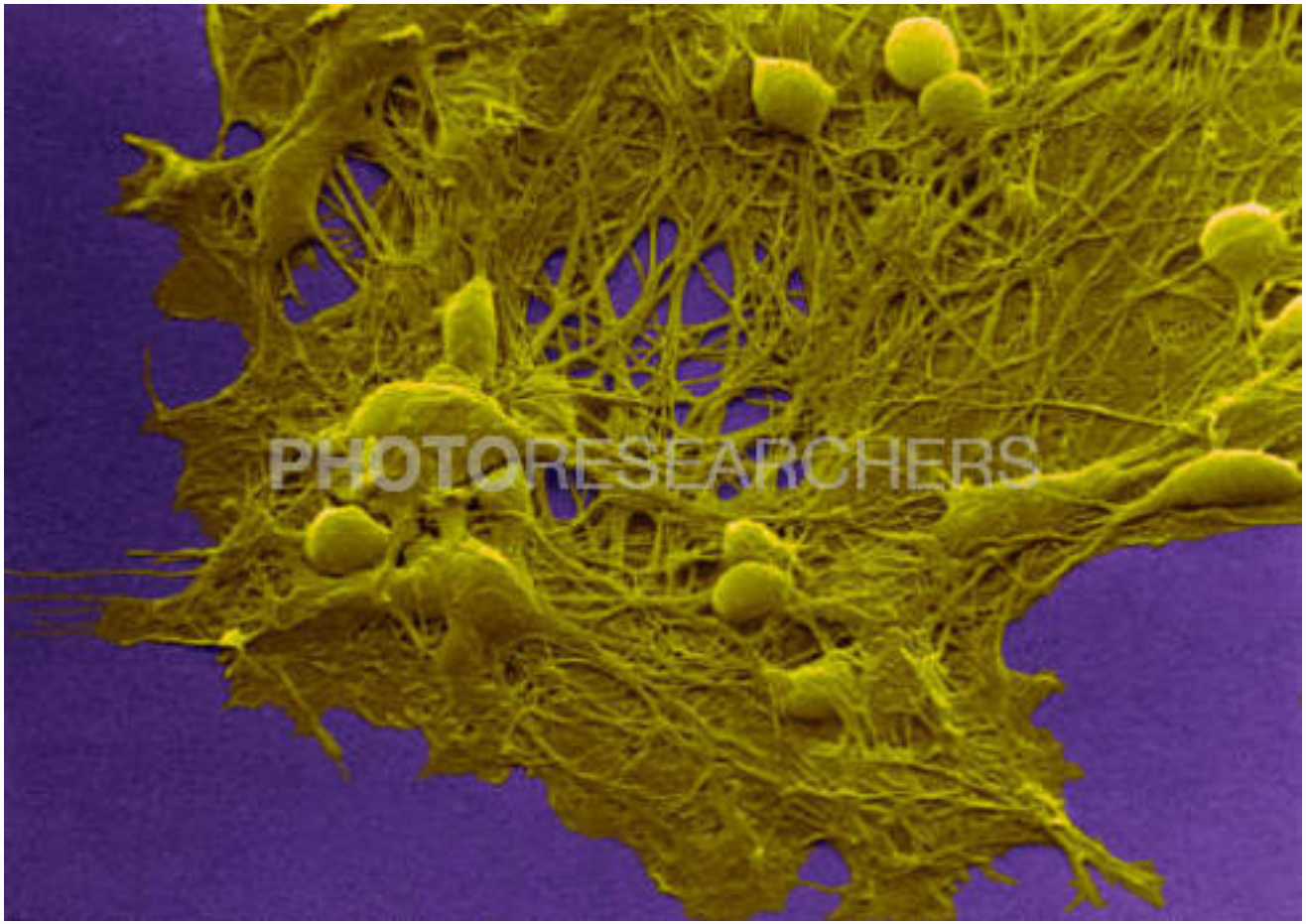
Spatiotemporal Processing



Plasticity

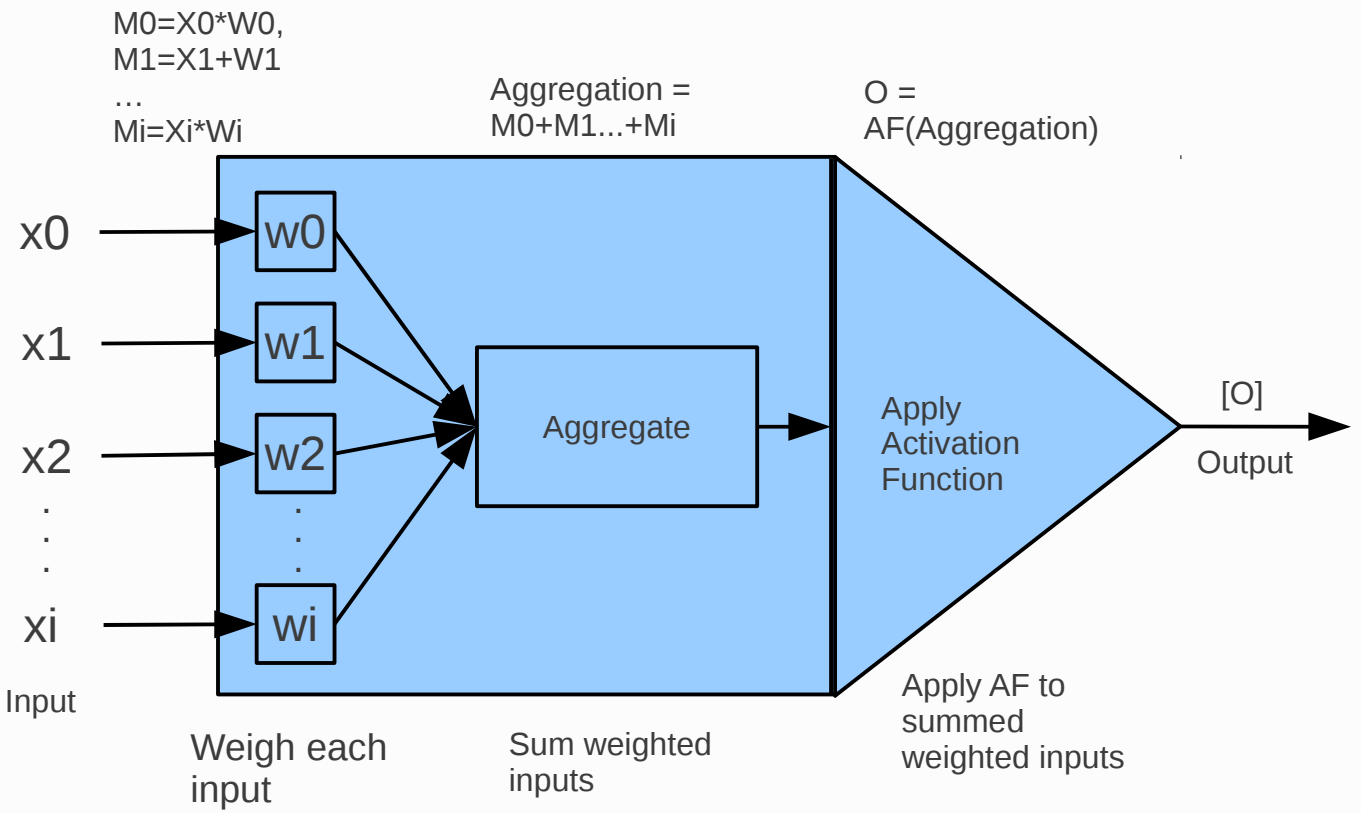
- Axon extension
- New dendrite branches
- More/less receptors
- Different signal integration properties due to shape and other changes
- ...

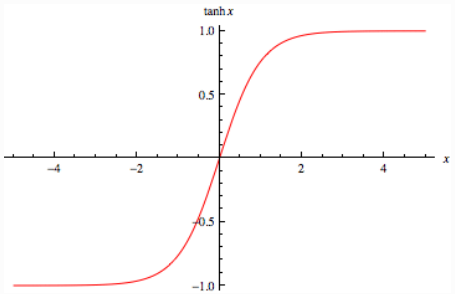
Putting it all together



Artificial Neural Network

Artificial neuron



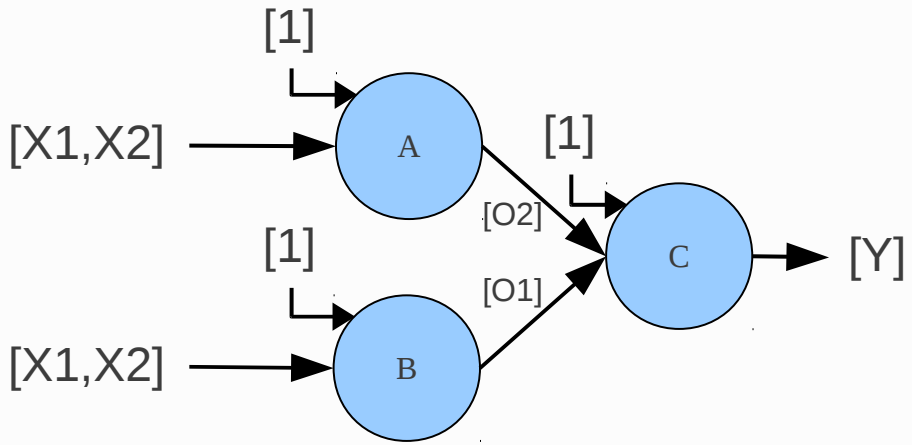


The Input is Just a Vector

1. Dot product:
DP=(0.5*-1) + (0.2*1)
Threshold = (0*1)

2. Activation strength:
Output = tanh(DP+Threshold)

Neural Circuit In Action



Input [X1,X2]: [-1,-1]
A: O1 = -0.9704 = tanh(-1*2.1081+ -1*2.2440 + 1*2.2533)
B: O2 = 0.9922= tanh(-1*3.4964 + -1*-2.7464 + 1*3.5200)
C: Y = **-0.99** = tanh(0.9922*-2.5983 + -0.9704*2.7354 + 1*2.7255)

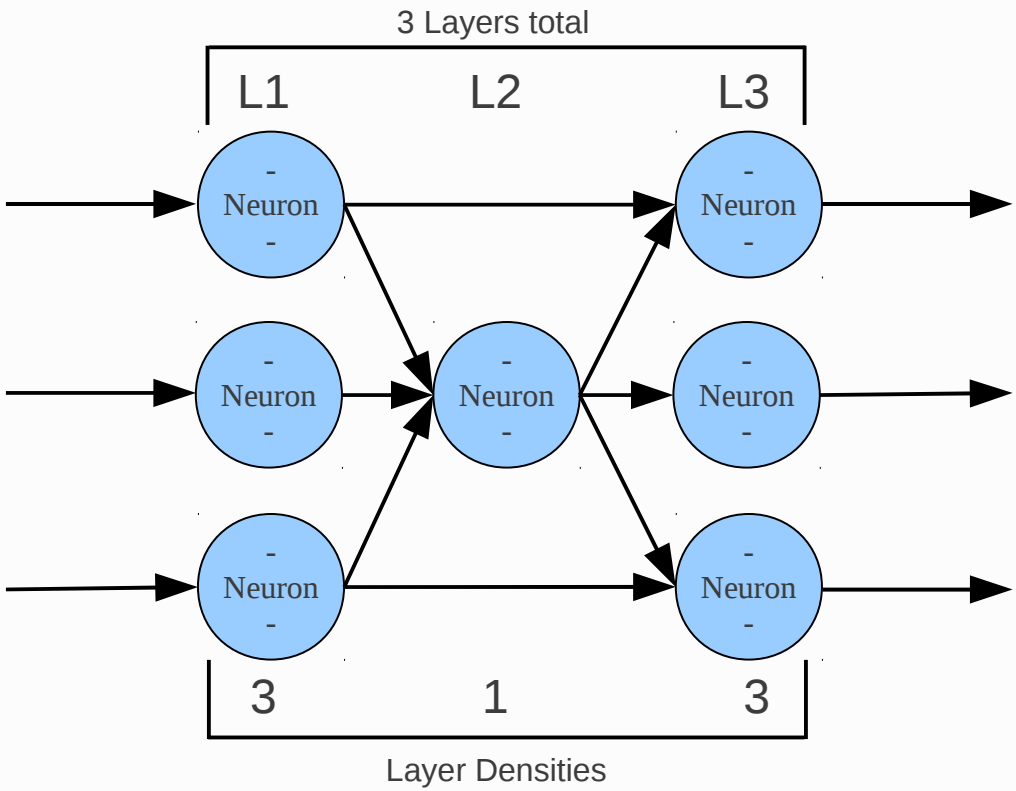
Input [X1,X2]: [-1,1]
A: O1 =0.9833 = tanh(-1*2.1081+ 1*2.2440 + 1*2.2533)
B: O2 = -0.9914= tanh(-1*3.4964 + 1*-2.7464 + 1*3.5200)
C: Y = **0.99** = tanh(-0.9914*-2.5983 + 0.9833*2.7354 + 1*2.7255)

Input [X1,X2]: [1,-1]
… C: Y = **0.99**

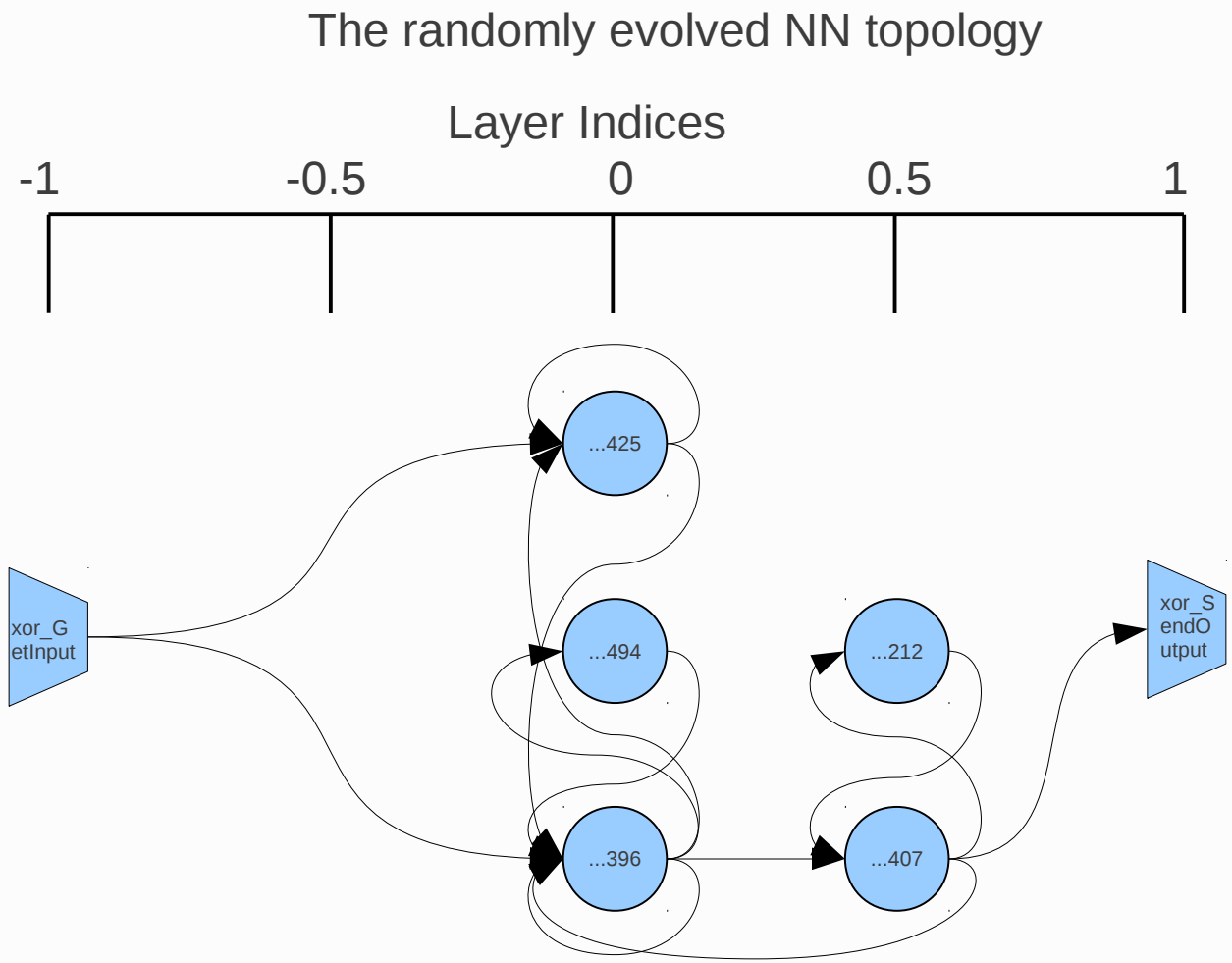
Input [X1,X2]: [1,1]
... C: Y = **-0.99**

$$Y = C(A(X1*Wa_1 + X2*Wa_2 + 1*Wa_3)*Wc_1 + B(X1*Wb_1 + X2*Wb_2 + 1*Wb_3)*Wc_2)$$

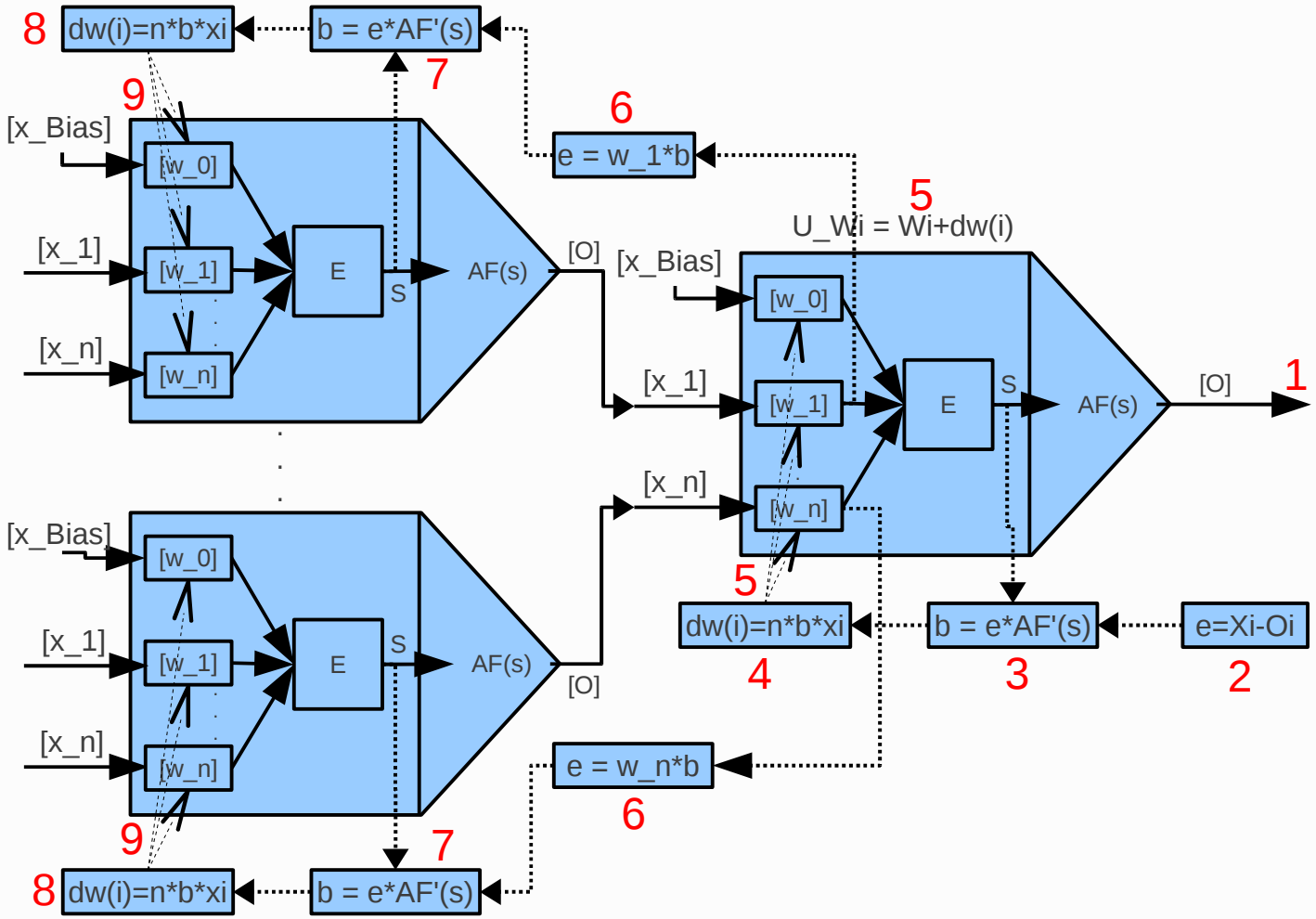
Neural Network



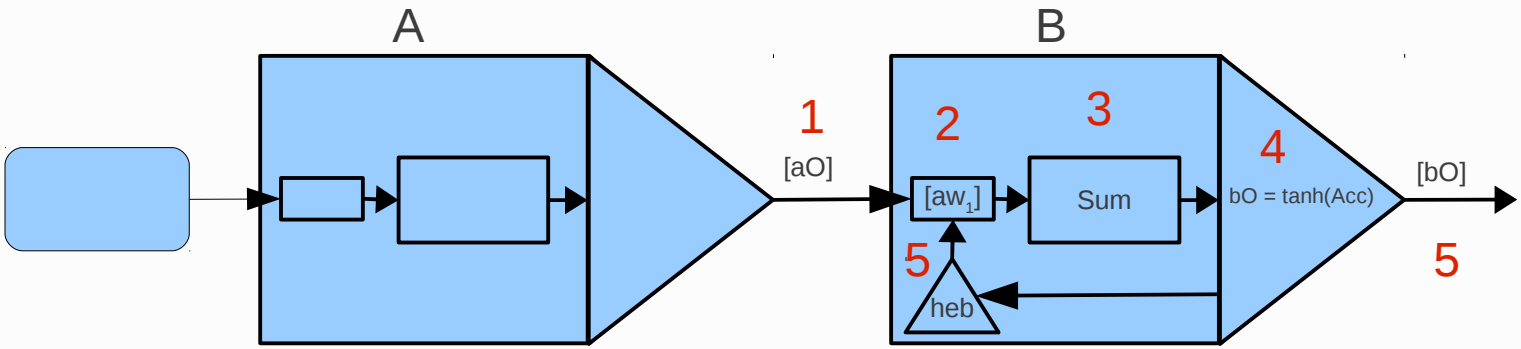
A Recurrent Neural Network



Error Backpropogation



Hebbian Learning



1. Neuron A sends the vector signal [aO] to B.
2. Signal aO is weighted with B's synaptic weight aw_1.
3. The weighted signals (in this case just one) are summed together to produce the value: Acc.
4. Activation function is applied to Acc to produce B's output signal bO.
5. B outputs vector signal [bO], while at the same time uses the Hebbian rule to produce a delta w, and update the synaptic weight aw_1.

Update Rule: $U\_W_i = W_i + n*X_i*O$
Where X_i is the presynaptic signal associated with synaptic weight W_i, and where O is the postsynaptic neuron's output, and n the learning parameter.

Example: aw_1 = 0.5, aO = 1, n = 1

1. Neuron A sends the vector signal [1] to B.
2. Signal 1 is weighted with B's synaptic weight 0.5 to produce $Y_1 = x_1*aw_1 = 1*0.5 = 0.5$.
3. The weighted signals (in this case just one, Y_1) are summed together: $Acc = Sum(Y_1) = 0.5$.
4. Activation function tanh is applied to Acc to produce B's output signal bO = tanh(Acc) = 0.46.
5. B outputs the vector signal [bO] = [0.46], while at the same time uses the Hebbian rule to produce: dw = 0.46 = 0.46*1, and update the synaptic weight aw_1. Thus, the updated aw_1 = 0.5 + 0.46 = 0.96. The new synaptic weight is: **aw_1 = 0.96**.

If we now continue running this update rule, with A firing signals of the same magnitude, 1, the sequence of B's weight aw_1 is: **0,5, 0.962, 1.71, 2.64, 3.63, 4.63**

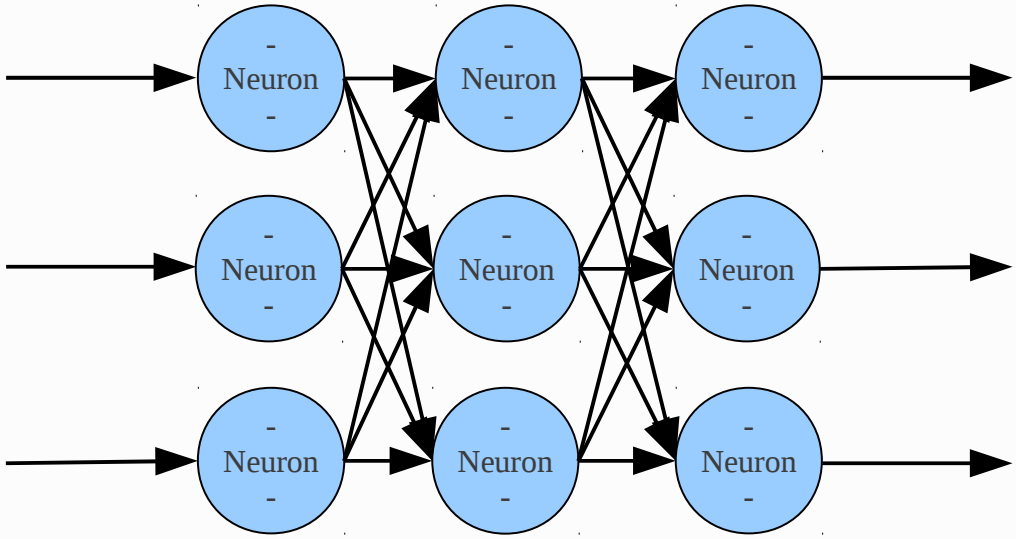
The synaptic weight continues to increase in magnitude over time.

Learning Vs. Training

- Supervised
  - Backpropagation
  - ...
- Unsupervised
  - Kohonan (Self-organizing) map
  - Adaptive Resonance Theory
  - Hebbian
    - "The general idea is an old one, that any two cells or systems of cells that are repeatedly active at the same time will tend to become 'associated', so that activity in one facilitates activity in the other." (Hebb 1949, p. 70)
  - Modulated
  - Evolutionary
  - ....

Putting it all together

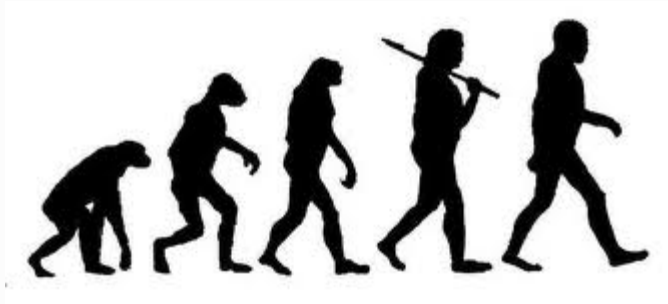
- Simulate biological NNs to various degrees of precision
- Directed graphs
- Parallel
- Learn, adapt, and generalize



Ok ok... But what about the topology, and the new learning parameters? How do I set them to the values that produce useful system for some problem?

Evolutionary Computation

- Based on evolutionary principles
- Stochastic search with a purpose
  - Create as many copies of yourself as possible
    - Some copies (offspring) will have errors when being copied
  - Others are competing for resources
  - Push towards finding an advantage
  - Survival of the fittest
- Genotype to Phenotype
- Mutation and crossover



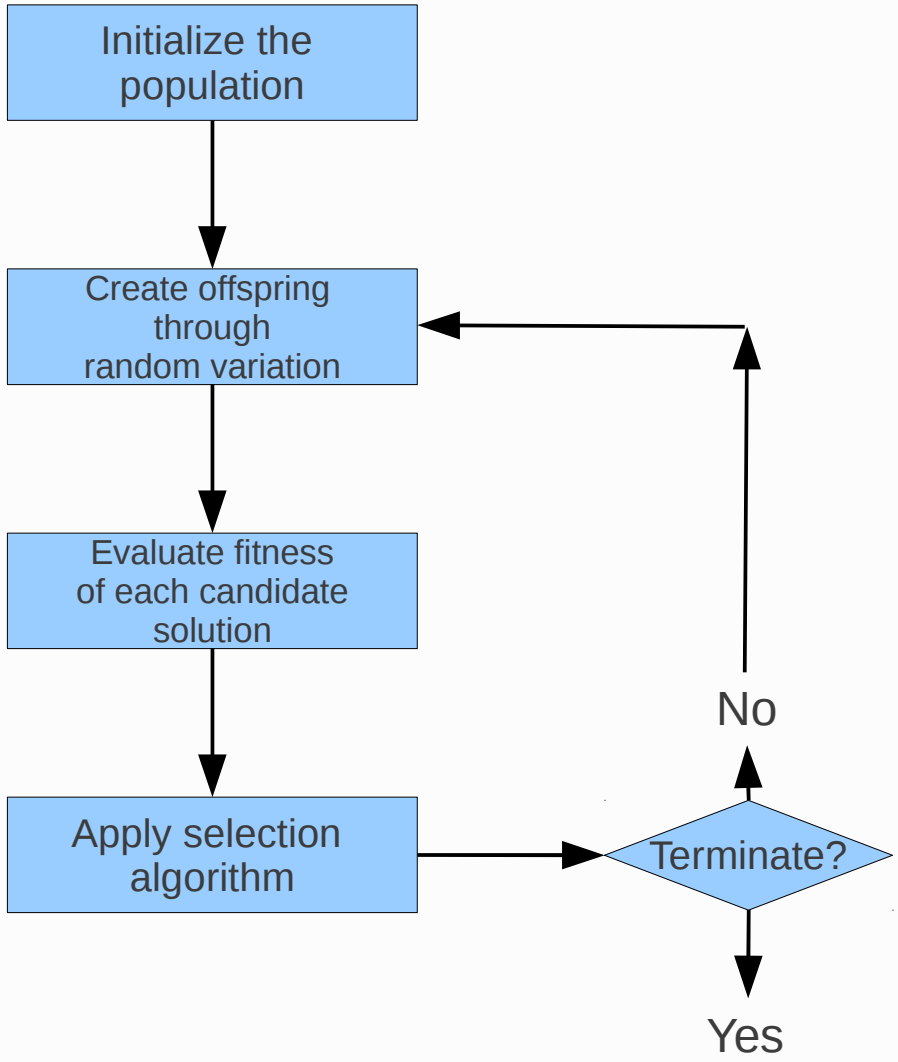
Evolution

- Start with inorganic molecules.
- Billions of planets, atoms and molecules hitting each other.
- Trillions of permutations, for billions of years, over billions of worlds, eventually one permutation will act in such a way that if some appropriate set of atoms is around, simply based on its chemical properties it will convert those resources into the same type molecule as itself. Replication on the most basic level.
- There could be no other way
- Replicators make copies of themselves.
- If you don't replicate, the replicators will use you as a resource to make copies of themselves.
- Copies are not perfect, there are errors.
- Most will be worse than the original replicators.
- Some will be better.
- Replicators interact with each other, beneficial errors (mutations), allow the agent to better compete with its own copies and others.
- Speciation occurs, those which can make more copies use more resources and overtake the environment... but sooner or later one of their copies will have a beneficial mutation...
- This is competition at the chemical level.
- The organisms compete, they make the environment more complex as they interact with each other and the environment.
- To deal with complex environment, they become more complex.
- An innovation: A replicator with a protective shell, a cell body, a survivor machine (Richard Dawkins – The Selfish Gene).
- Result: an adaptive survival machine, an agent that is capable of surviving in a dynamic and hostile environment

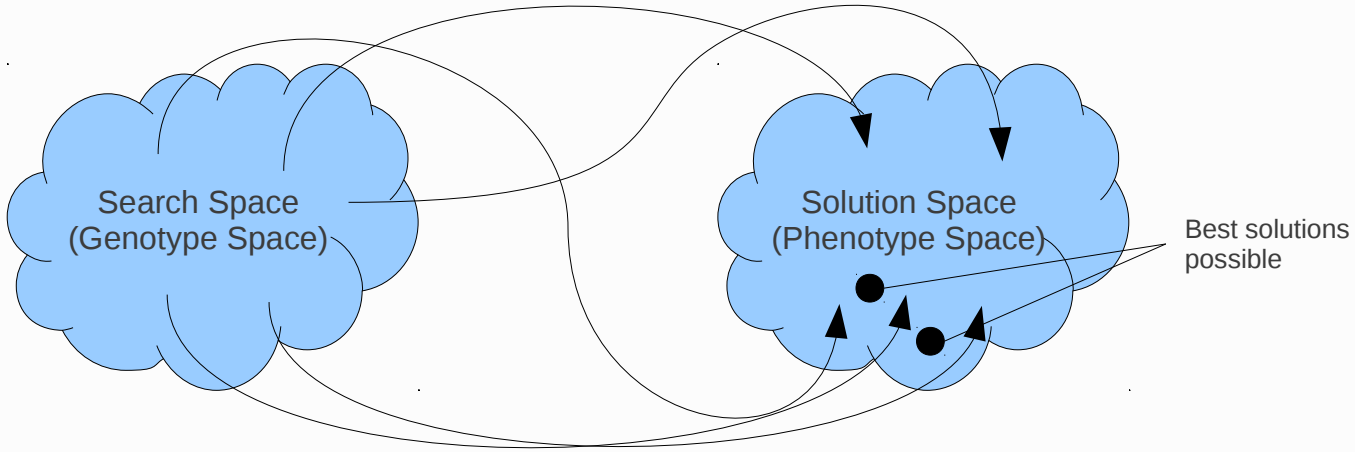
Evolutionary Computation Flowchart

Extracting the most important parts:

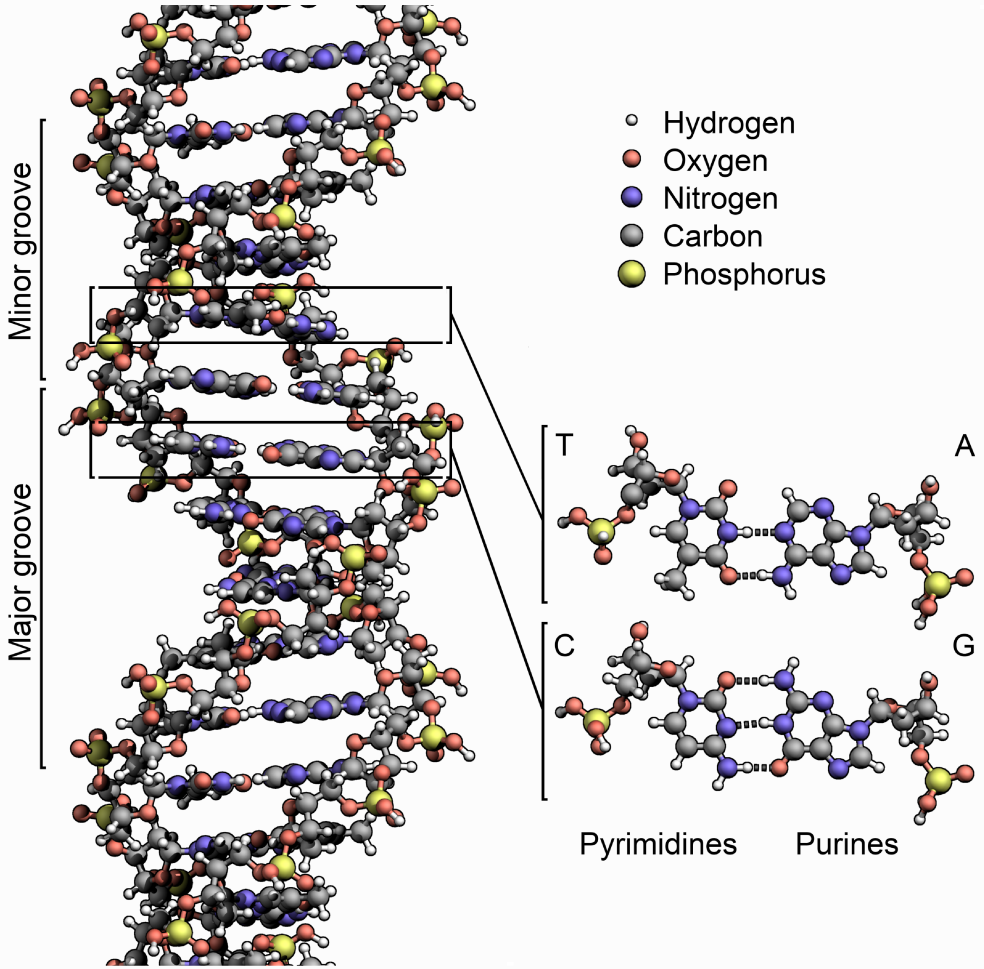
1. Replication.
2. Variation: Mutation.
3. Competition: Those that are more fit, will survive and make more mutant copies of themselves.

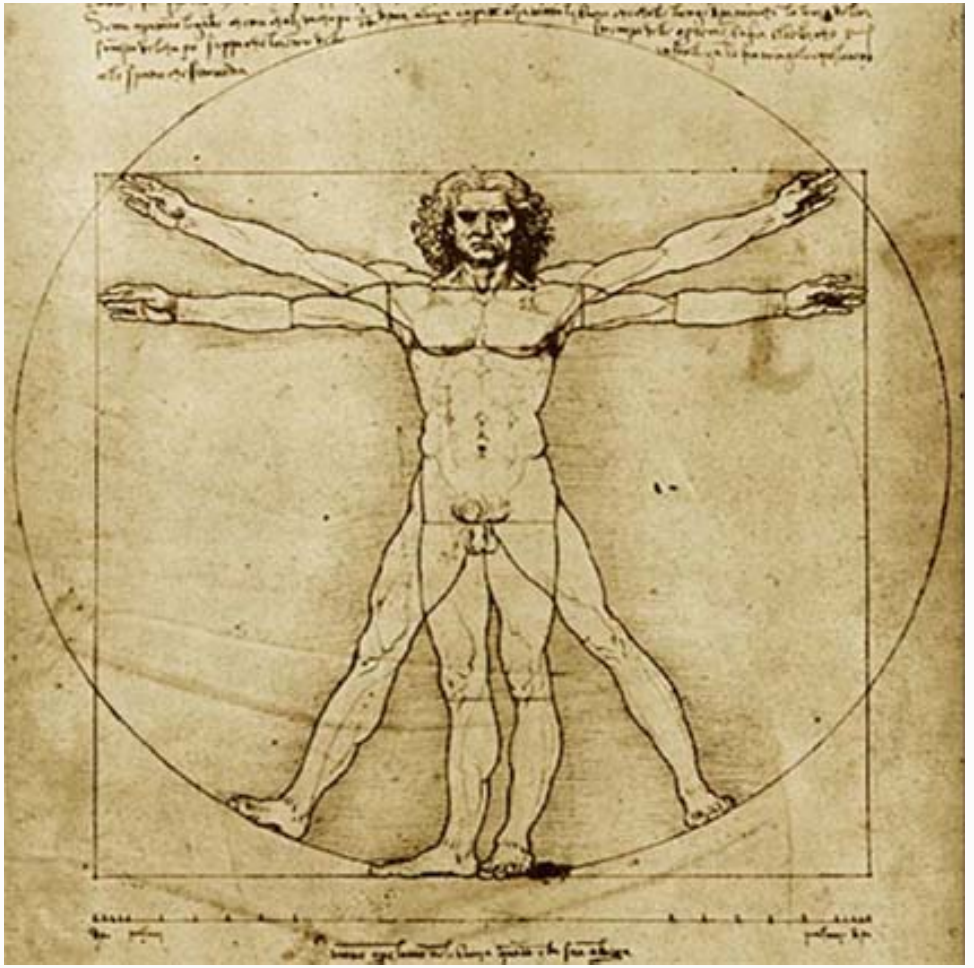


Genotypes and Phenotypes



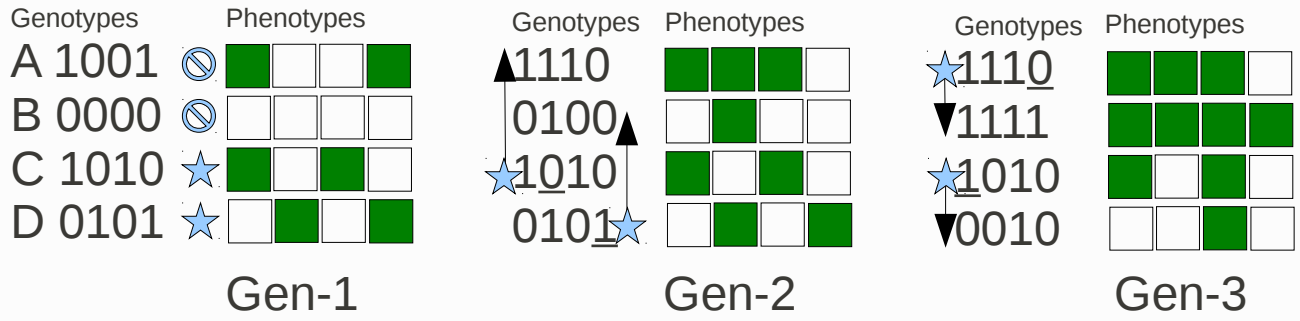
DNA → (RNA → Protien) → Organism



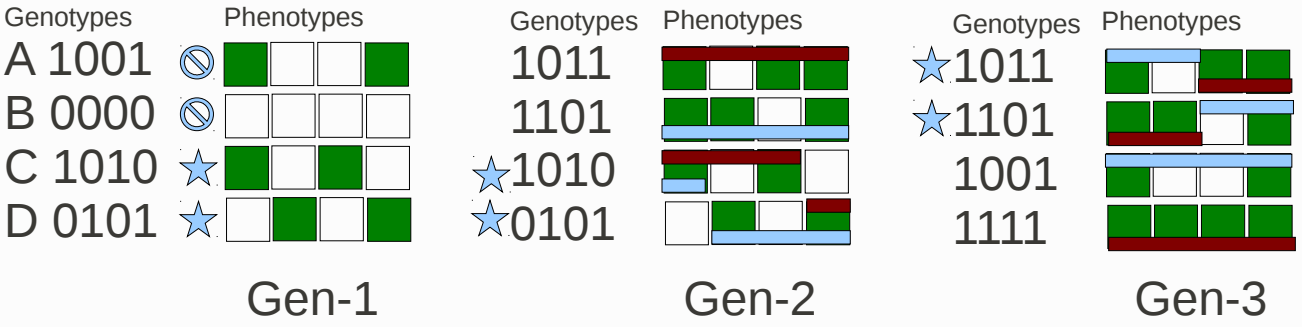


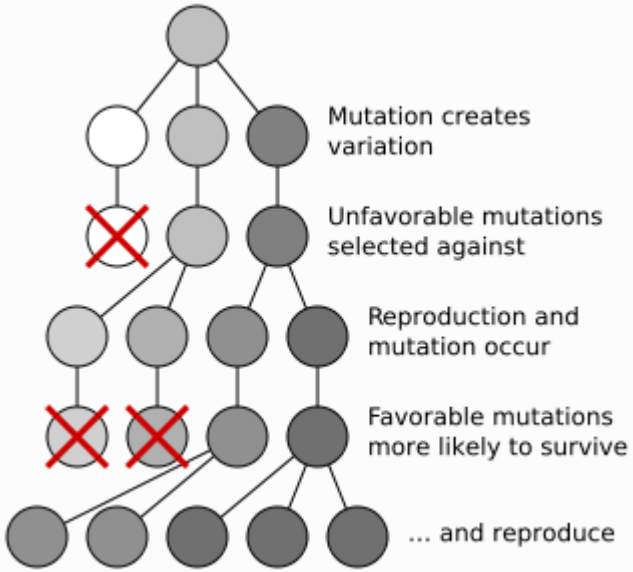
Simple Genetic Algorithm Example

Simple Mutations



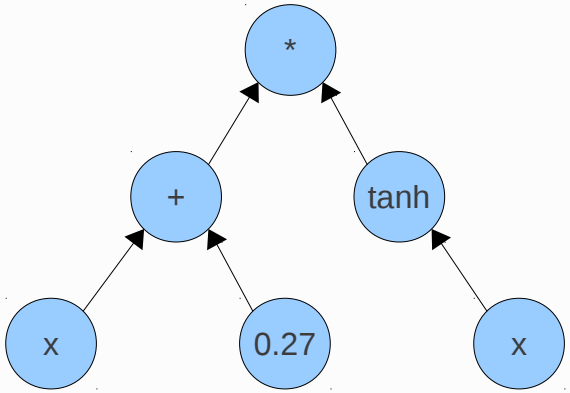
Crossover





Genetic Programming

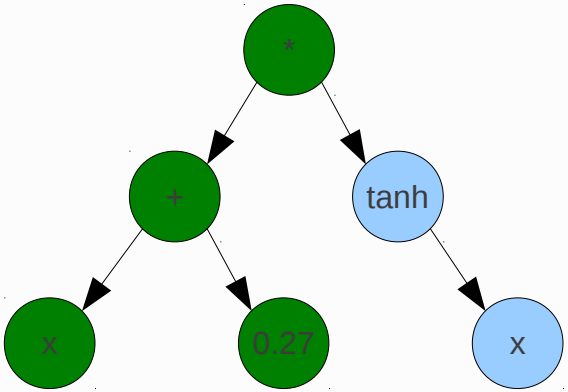
Tree encoded genotype:



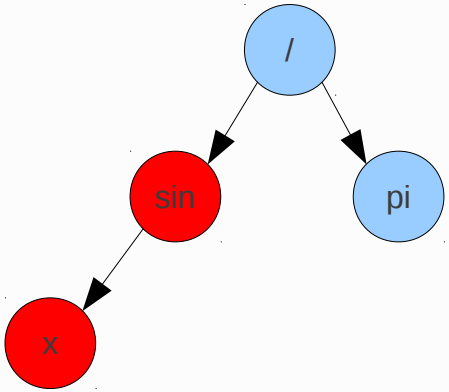
Phenotype: (x+0.27)*tanh(x)

Genetic Programming

Agent: A

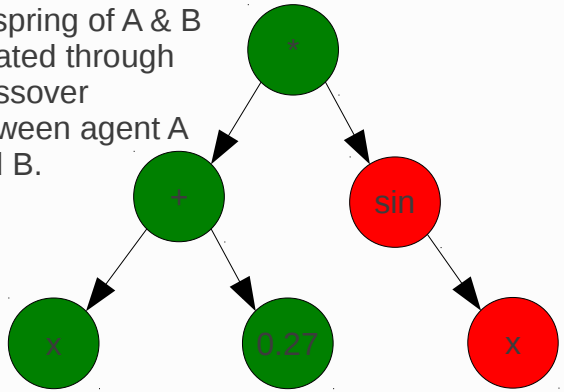


Agent: B



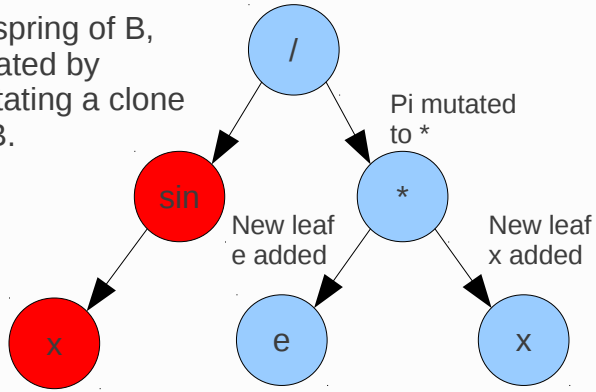
Agent: C

Offspring of A & B created through crossover between agent A and B.



Agent: D

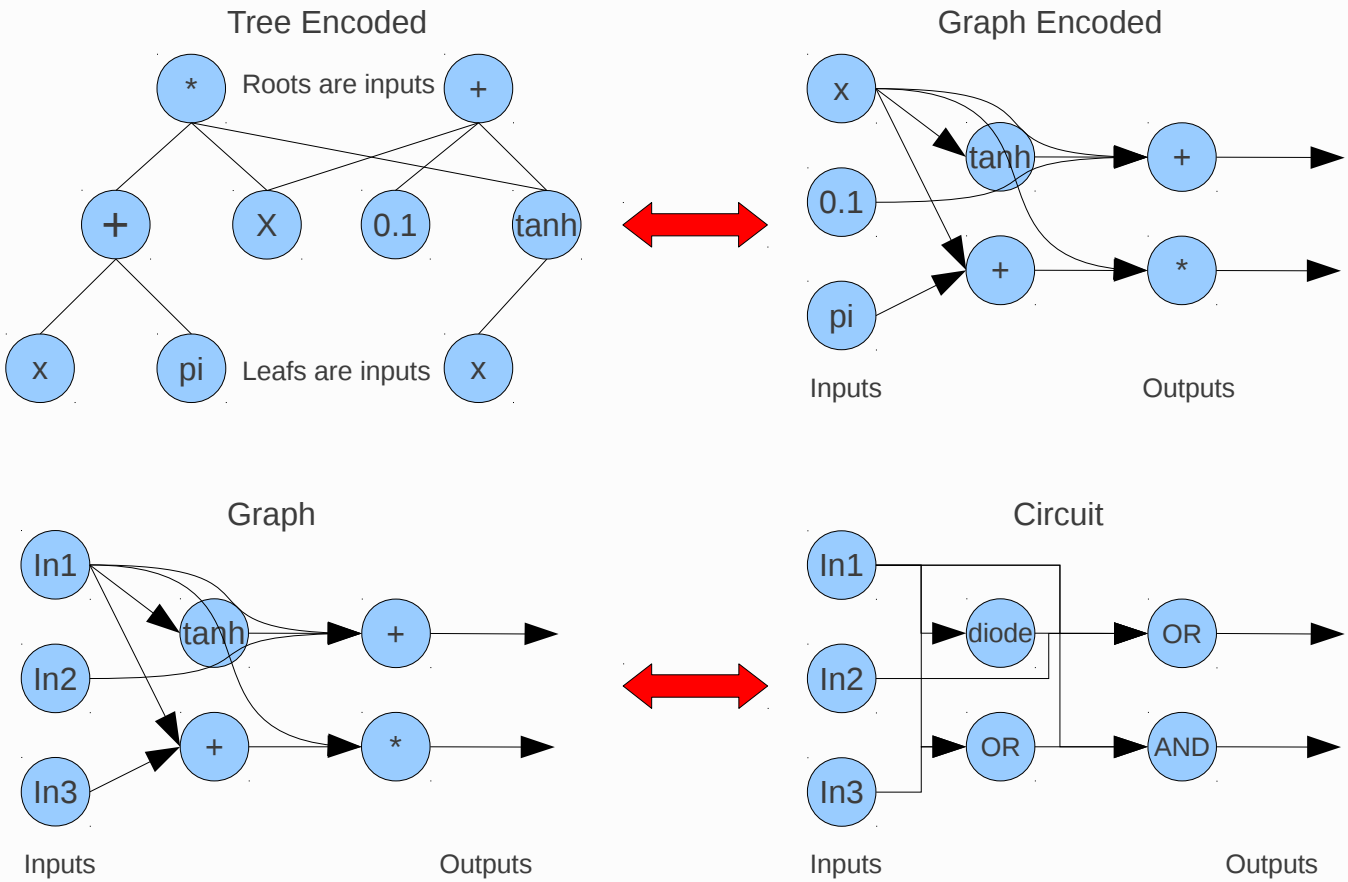
Offspring of B, created by mutating a clone of B.



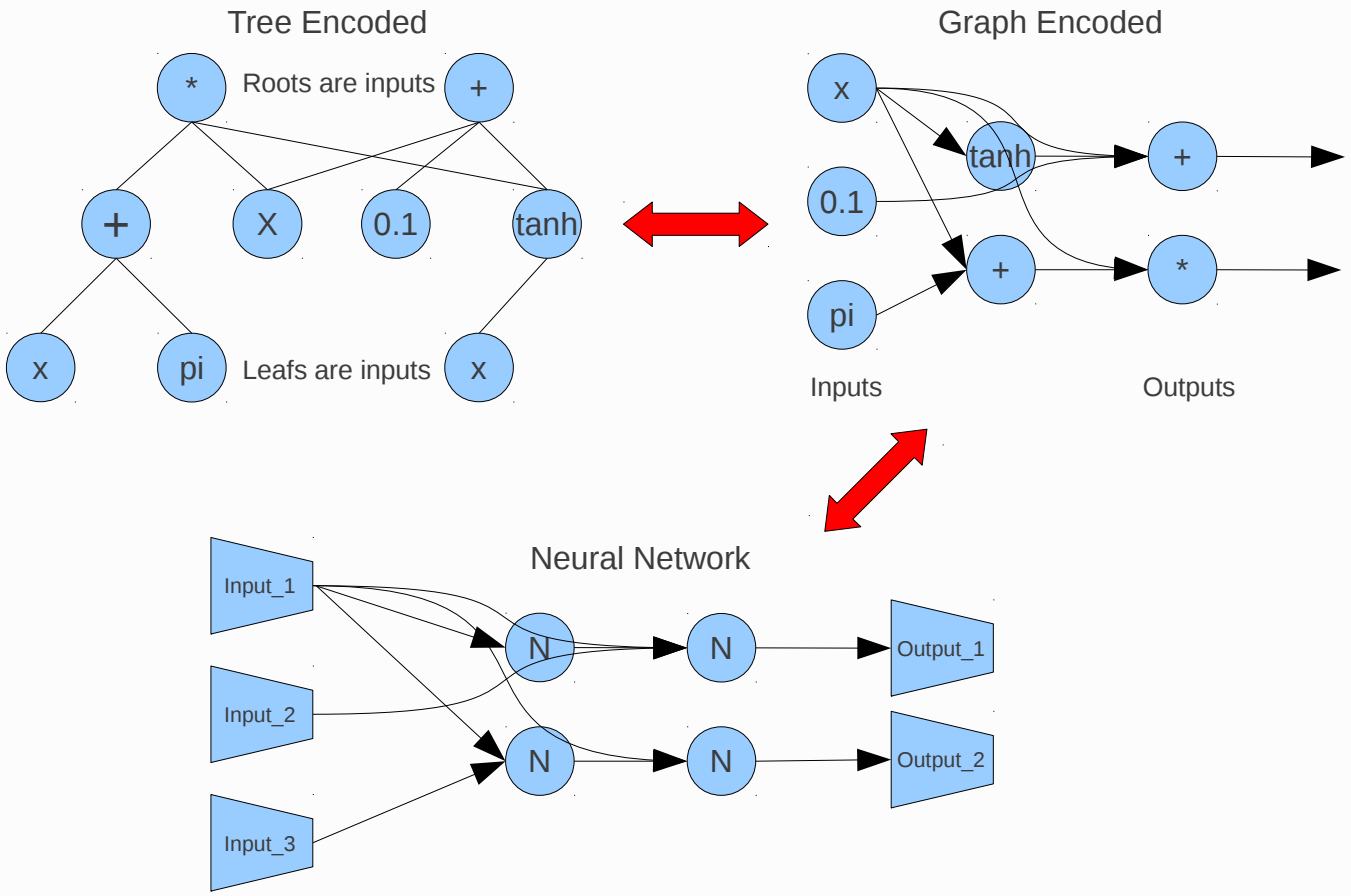
Evolutionary Computation Approaches

- Genetic Algorithms (John Holland, 73-75)
  - Population of fixed length genotypes, bit strings, evolved through perturbation/crossing
- Genetic Programming (John Koza, 92)
  - Variable sized chromosome based programs represented as treelike structures, with specially crafted genetic operators
- Evolutionary Strategies (Ingo Rechenberg, 73)
  - Normal distribution based, adaptive perturbations (self-adaptation)
- Evolutionary Programming (L. & D. Fogel, 63)
  - Like ES, but for evolution of state transition tables for finite-state machines (FSMs)

Different sides of the same coin

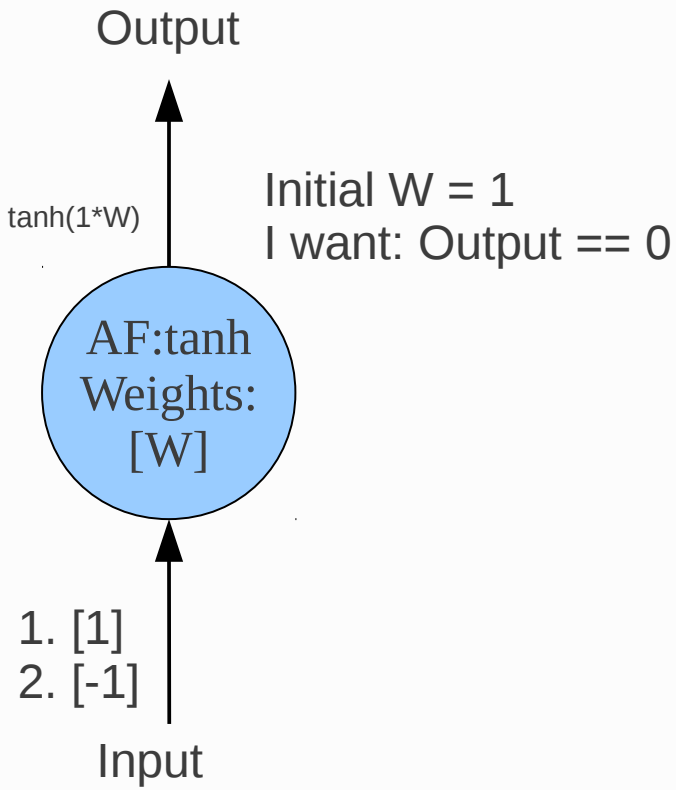


Towards Neuroevolution



Neuroevolution

Stochastic Hill Climber



1. Output1 = tanh(1*1) = 0.76
   Output2 = tanh(1*-1) = -0.76

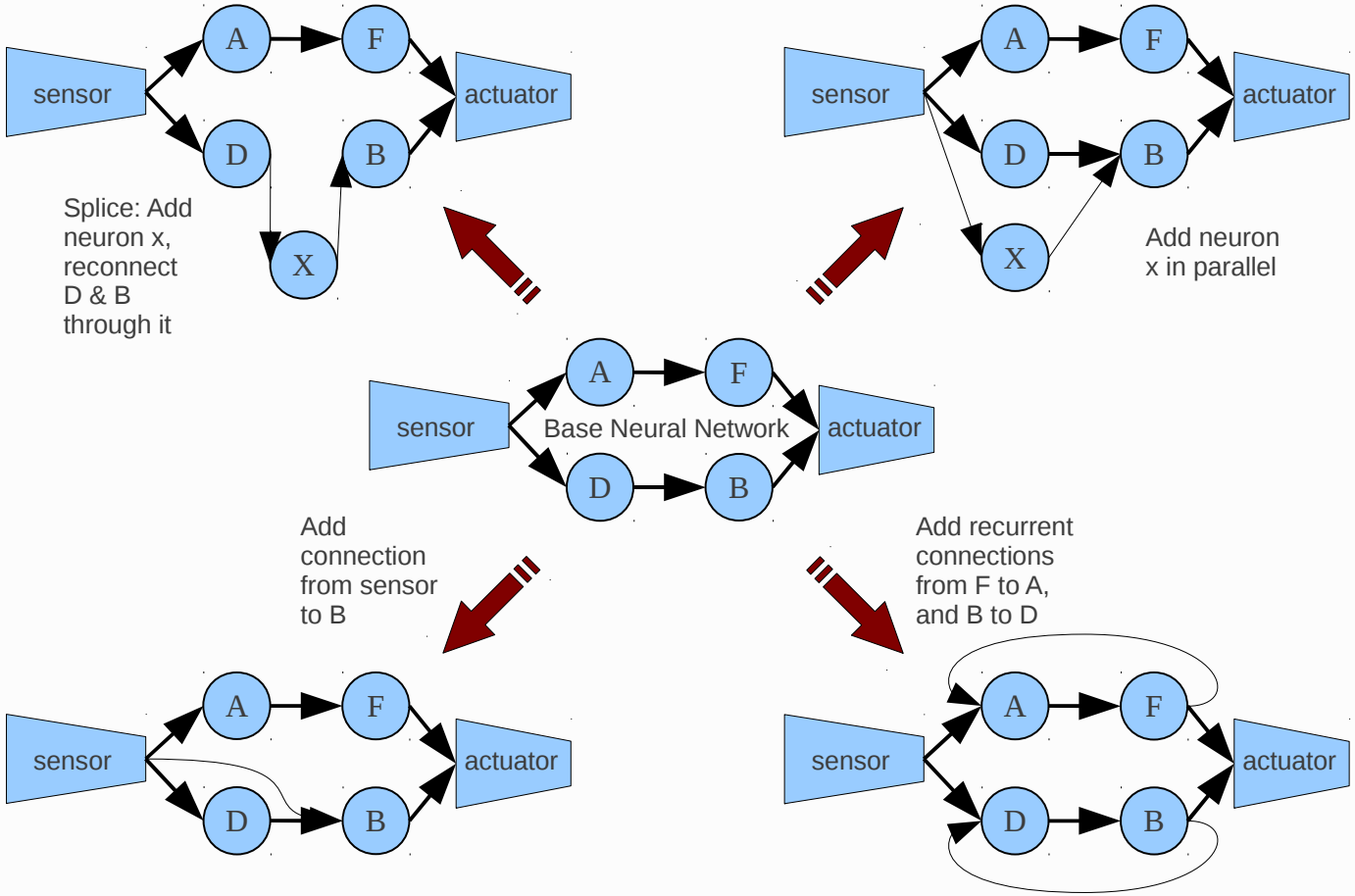
2. Weight Perturbation
Perturbation = -0.5
Try W = 0.5 = 1 - 0.5
Output1 = tanh(0.5*1) = 0.46
Output2 = tanh(0.5*-1) = -0.46
That's closer! New W = 0.5

3. Weight Perturbation
Perturbation = +0.2
Try W = 0.7 = 0.5 + 0.2
Output1 = tanh(0.7*1) = 0.60
Output2 = tanh(0.7*-1) = -0.60
Not as good as before, New W = 0.5

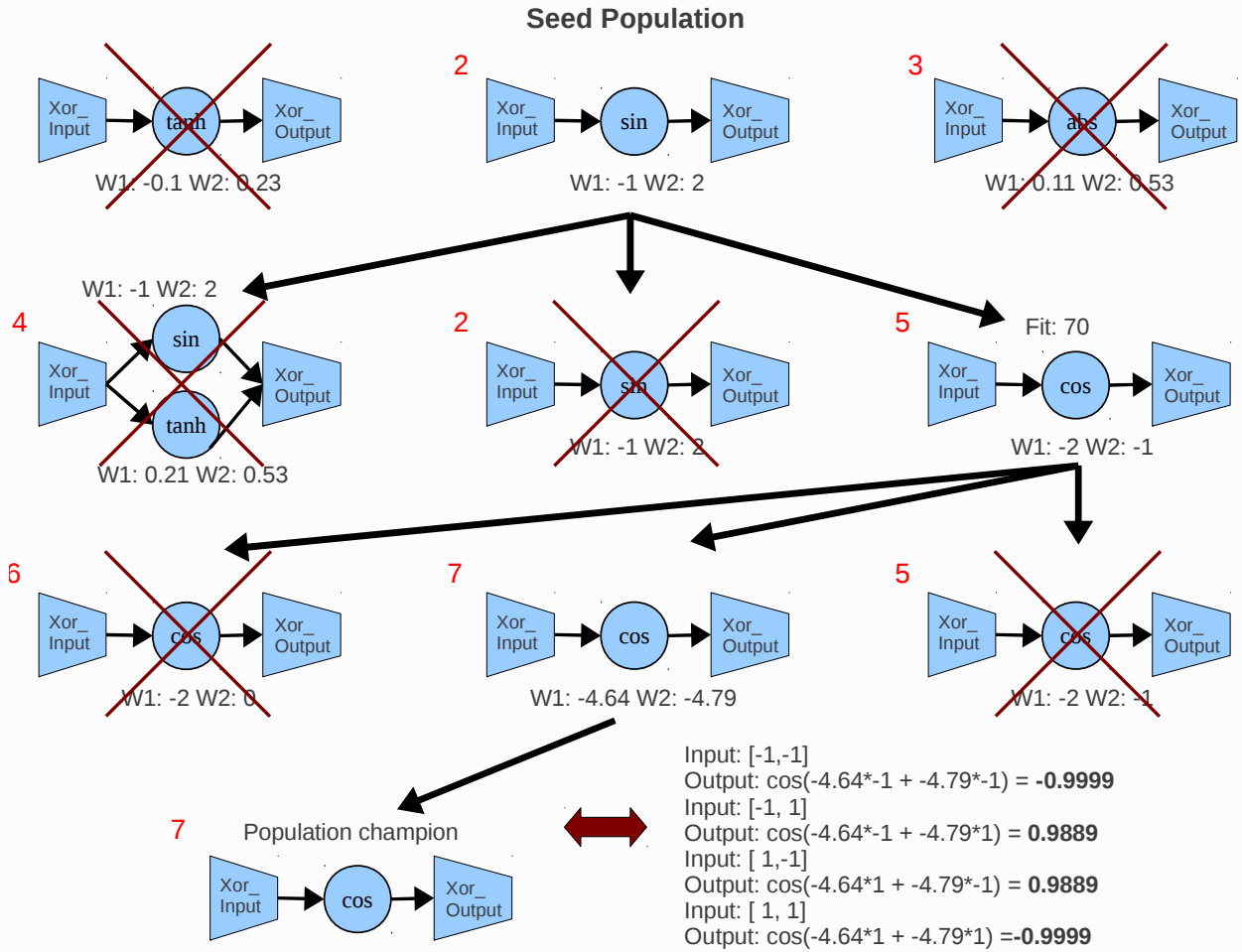
4. Weight Perturbation
Perturbation = -0.5
Try W = 0 = 0.5 - 0.5
Output1 = tanh(0*1) = 0 !!!
Output2 = tanh(0*-1) = 0 !!!

The right weight is 0.

What about topological mutation operators



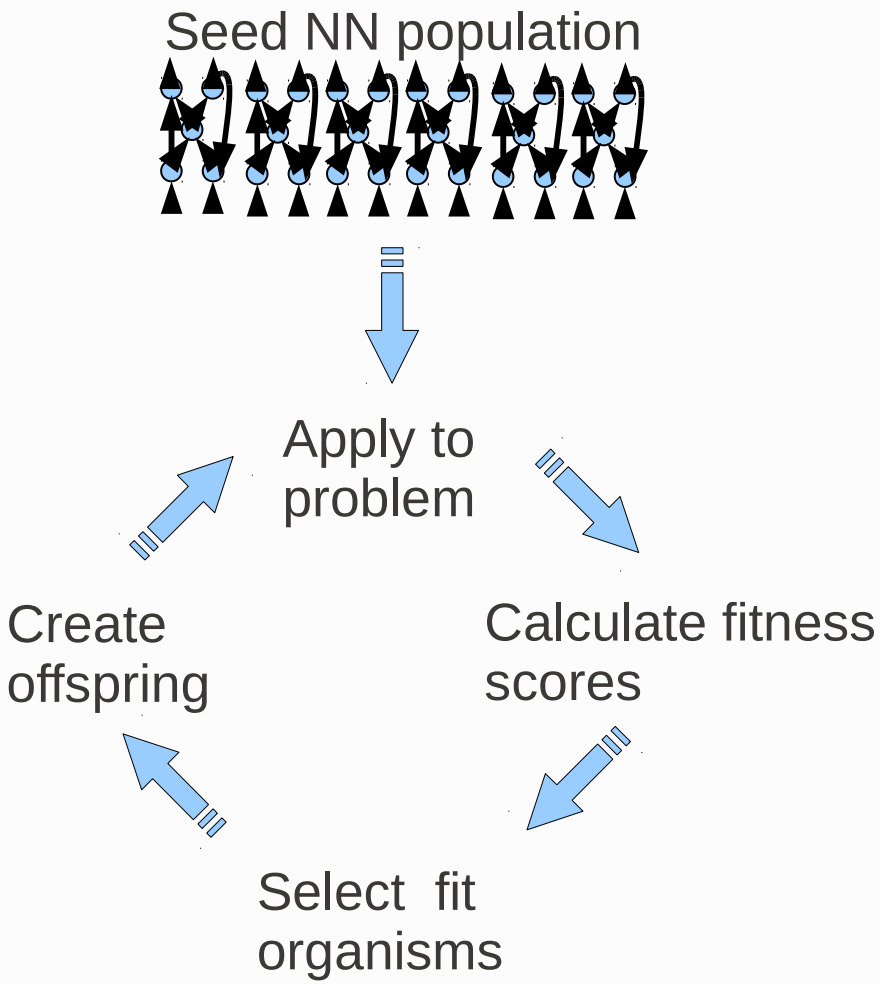
Instead of evolving a single NN, let's evolve a population



Topology and Weight Evolving Artificial Neural Networks

- Populations and Fitness Functions
- Parametric mutation operators
- Topological mutation operators
- Other mutation operators
  - Learning Algorithms
  - Activation Functions
  - Crystalization
  - Full genome duplication
  - …

TWEANN Cycle



Erlang: The Quintessential Neural Network Programming Language

The Unintentional Neural Network Programming Language

”A list of features that a neural network based computational intelligence system needs, as quoted from the list made by Bjarne Däcker [1], is as follows:

1. The system must be able to handle very large numbers of concurrent activities.
2. Actions must be performed at a certain point in time or within a certain time.
3. Systems may be distributed over several computers.
4. The system is used to control hardware.
5. The software systems are very large.
6. The system exhibits complex functionality such as, feature interaction.
7. The systems should be in continuous operation for many years.
8. Software maintenance (reconfiguration, etc) should be performed without stopping the system.
9. There are stringent quality, and reliability requirements.
10. Fault tolerance

Surprisingly enough, Däcker was not talking about a neural network based general computational intelligence systems when he made this list, he was talking about a telecom switching systems.”

[1] Bjarne Däcker. Concurrent functional programming for telecommunications: A case study of technology introduction. November 2000. Licentiate Thesis.

Necessary Features for a NN-PL

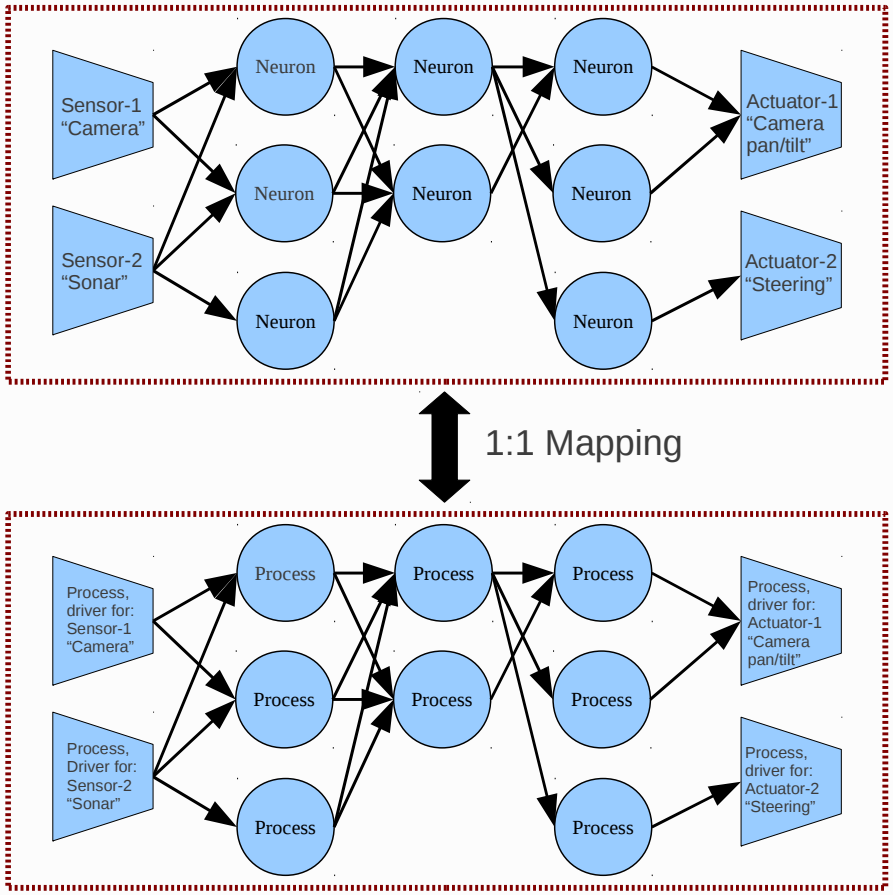
(These will sound very familiar)

- Encapsulation
- Concurrency of Neuron primitives
- Fault detection primitives
- Location transparency
- Dynamic code upgrade

Erlang's Features

- Encapsulation primitives
- Concurrency
- Fault detection primitives
- Location transparency
- Dynamic code upgrade

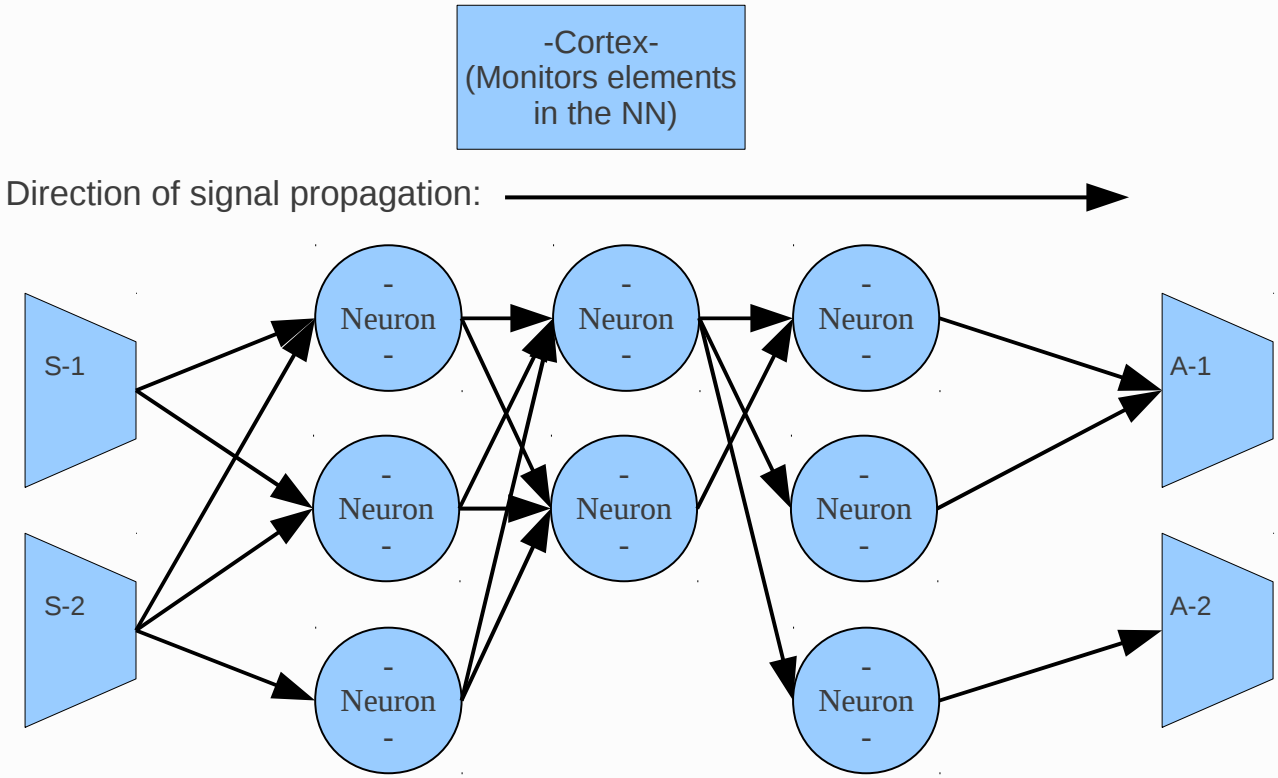
The Topological 1:1 Mapping



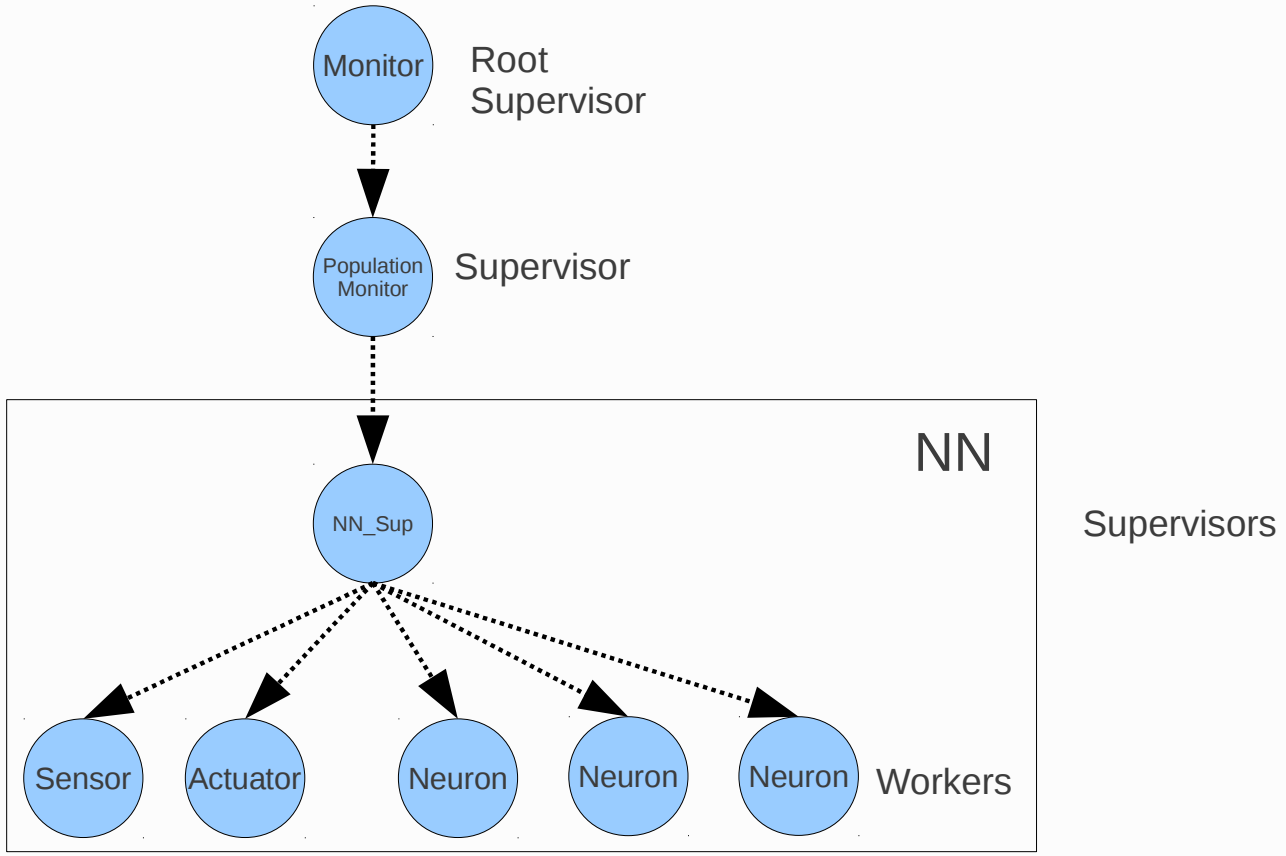
Neural Network Based Computational Intelligence System, interfacing with a robotic body

Representation of the system in Erlang

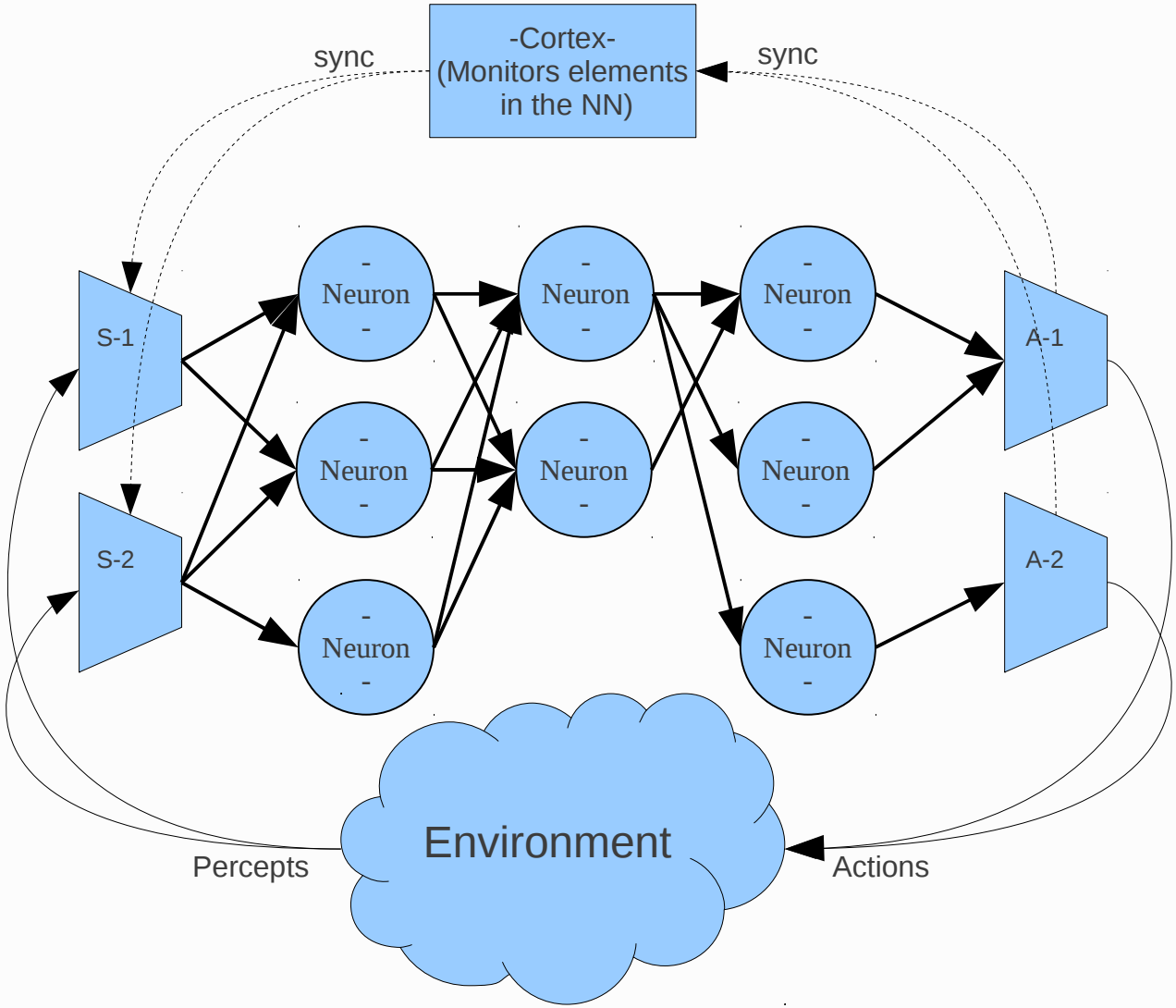
A simple neural network



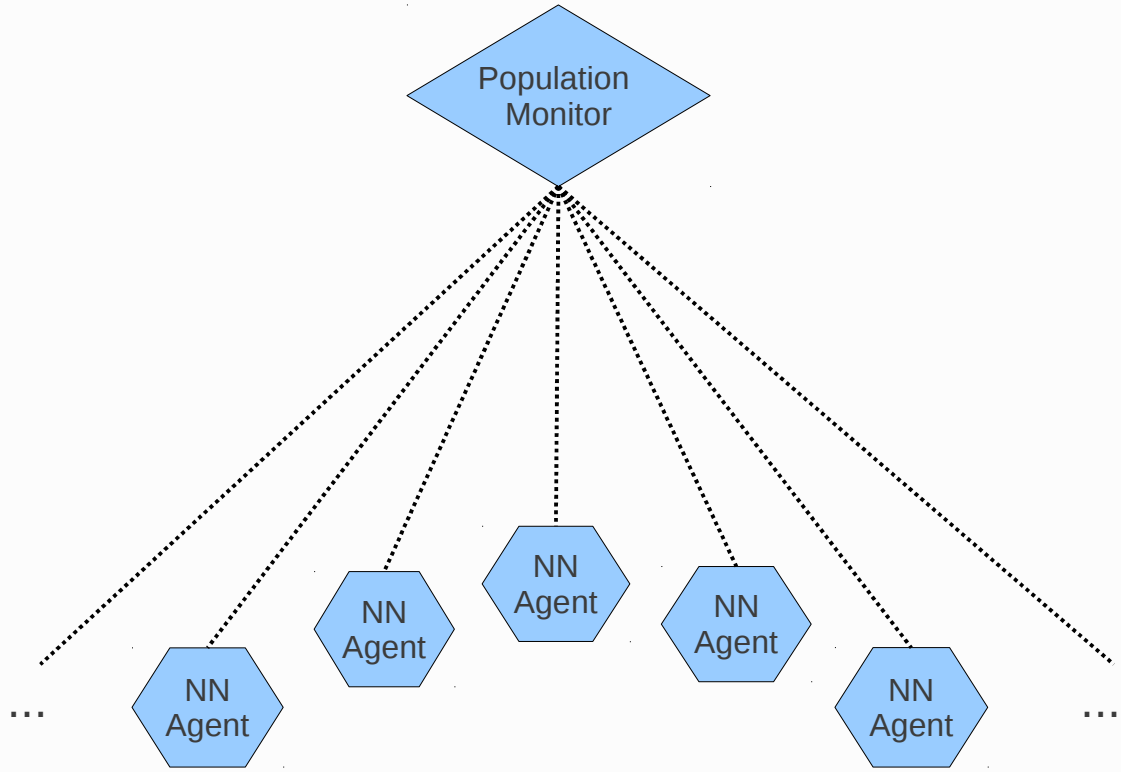
Monolithic NN Supervision Tree



The architecture of an Infomorph

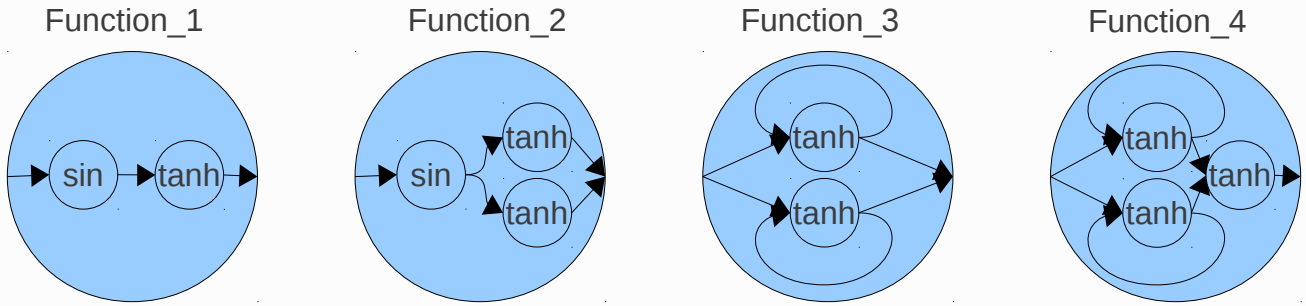


An evolving NN population

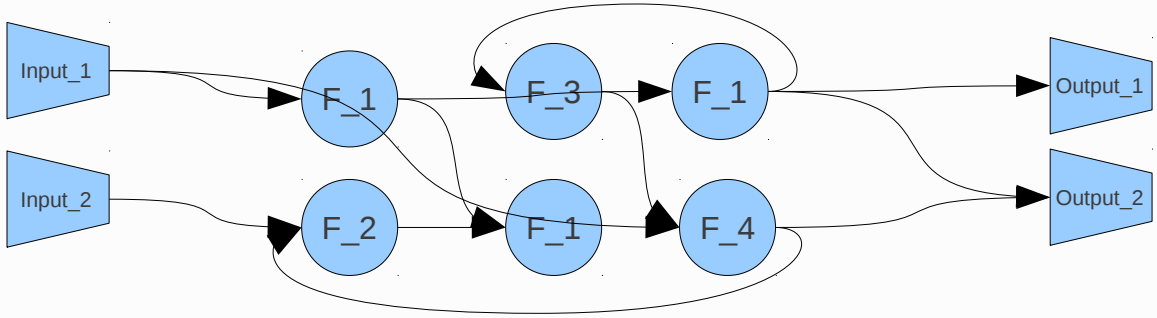


Groups of Neural Circuits

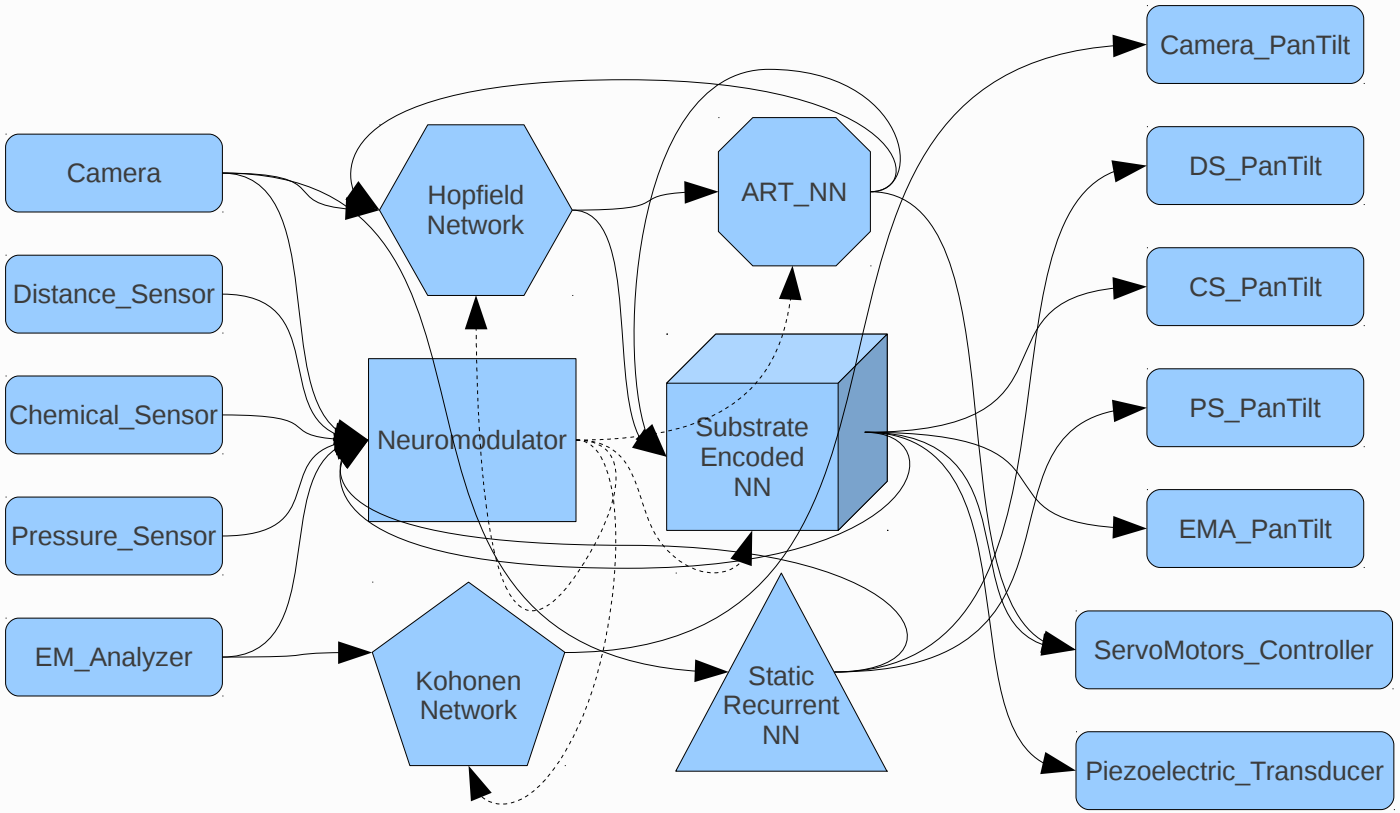
Various functions created through neural circuits, which are universal function approximators



The following Neural Network is composed from the above neural circuits. Since any function can be created by a neural circuit, the NNs are just as flexible as a genetic programming based system.



Modular NNs


Camera
Distance_Sensor
Chemical_Sensor
Pressure_Sensor
EM_Analyzer
Hopfield Network
Neuromodulator
Kohonen Network
ART_NN
Substrate Encoded NN
Static Recurrent NN
Camera_PanTilt
DS_PanTilt
CS_PanTilt
PS_PanTilt
EMA_PanTilt
ServoMotors_Controller
Piezoelectric_Transducer

What Erlang Offers to the Field of NN Research

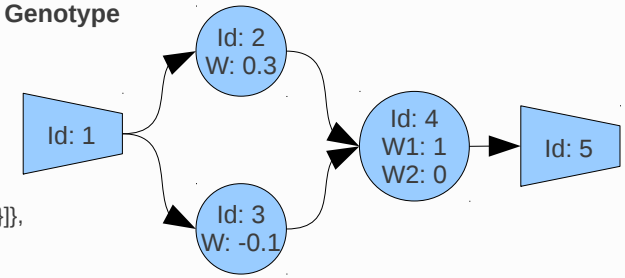
- Augmenting topologies live
- Full distribution and utilization of hardware
- Fault tolerance; ”stroke recovery”
- 1:1 mapping, from ideas to prototype systems
  - No need to overcome linguistic determinism
- Switching and adding new modules, no matter what they do, can be done live with code hotswappig
- Flexibility... in everything.

DXNN: A Case Study

Genotype Example

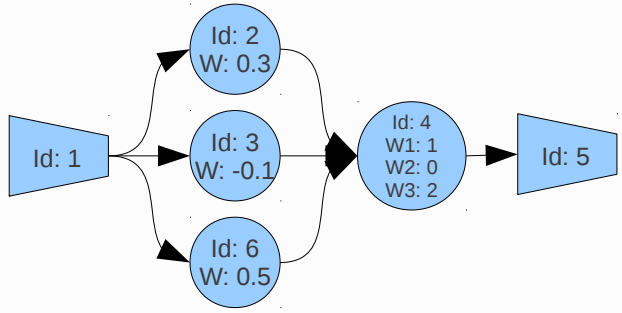
Initial Neural Network Genotype

```
{NN_System_Id=nn1,[
  {id={sensor_function_name,1},
     input_IdPs=void,output_Ids=[{neuron,2},{neuron,3}]},
  {id={neuron,2},
     input_IdPs=[{{sensor,1},0.3}],output_Ids=[{neuron,4}]},
  {id={neuron,3},
     input_IdPs=[{{sensor,1},-0.1}], output_Ids=[{neuron,4}]},
  {id={neuron,4},
     input_idps=[{{neuron,2},1},{{neuron,3},0}],output_ids=[{actuator,5}]},
  {id={actuator_function_name,5},
    input_idps=[{neuron4}], output_ids=void}]}.
```



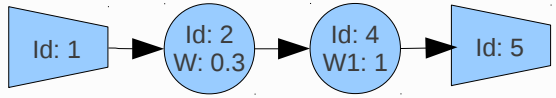
After adding element: {neuron,6} to the initial genotype

```
{NN_System_Id=nn1,[
  {id={sensor_function_name,1},
     input_IdPs=void,output_Ids=[{neuron,2},{neuron,3},{neuron,6}]},
  {id={neuron,2},
     input_IdPs=[{{sensor,1},0.3}],output_Ids=[{neuron,4}]},
  {id={neuron,3},
     input_IdPs=[{{sensor,1},-0.1}], output_Ids=[{neuron,4}]},
  {id={neuron,4},
     input_idps=[{{neuron,2},1},{{neuron,3},0},{{neuron,6},2}],
     output_ids=[{actuator,5}]},
  {id={neuron,6},
     input_idps=[{{sensor,1},0.5}],output_ids=[{neuron,4}]},
  {id={actuator_function_name,5},
    input_idps=[{neuron4}], output_ids=void}]}.
```



After removing element: {neuron,3} from the initial genotype

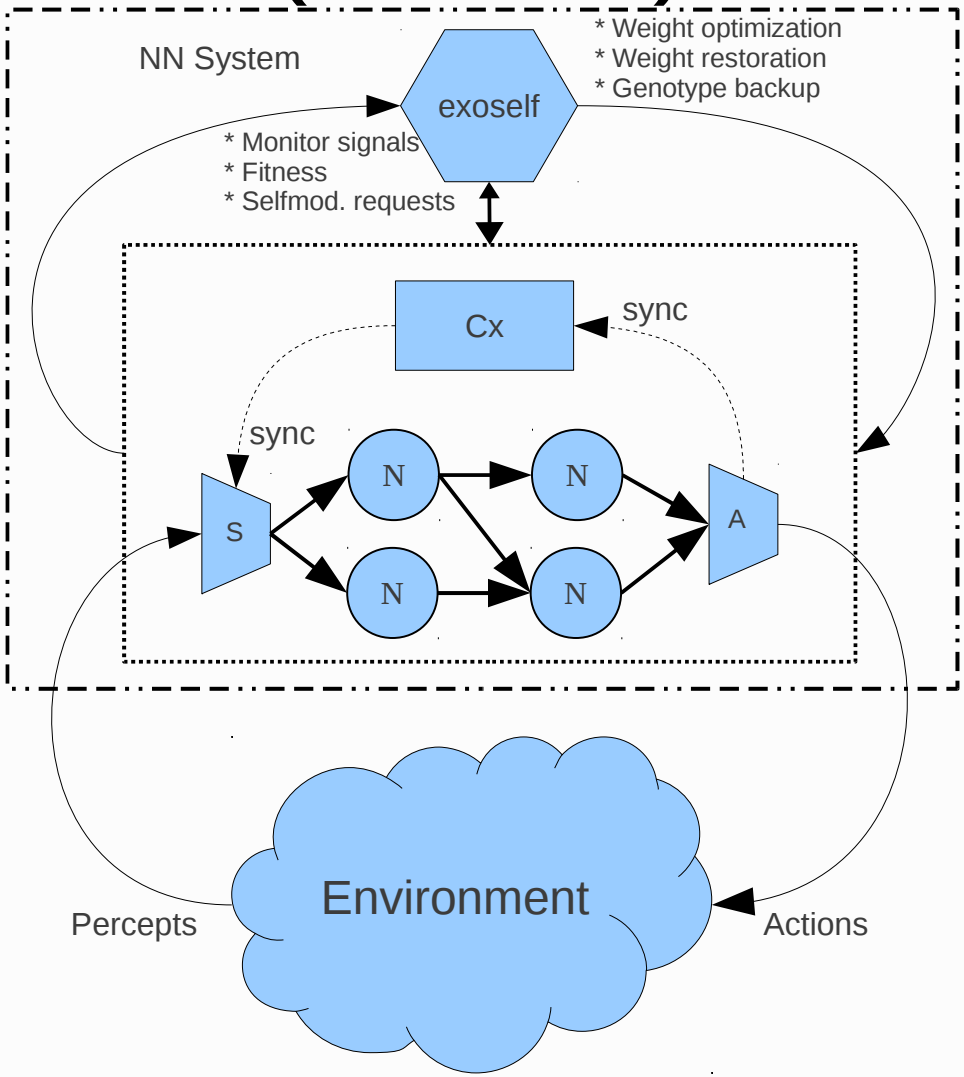
```
{NN_System_Id=nn1,[
  {id={sensor_function_name,1},
     input_IdPs=void,output_Ids=[{neuron,2},{neuron,3}]},
  {id={neuron,2},
     input_IdPs=[{{sensor,1},0.3}],output_Ids=[{neuron,4}]},
  {id={neuron,3},
     input_IdPs=[{{sensor,1},-0.1}], output_Ids=[{neuron,4}]},
  {id={neuron,4},
     input_idps=[{{neuron,2},1},{{neuron,3},0}],output_ids=[{actuator,5}]},
  {id={actuator_function_name,5},
    input_idps=[{neuron4}], output_ids=void}]}.
```



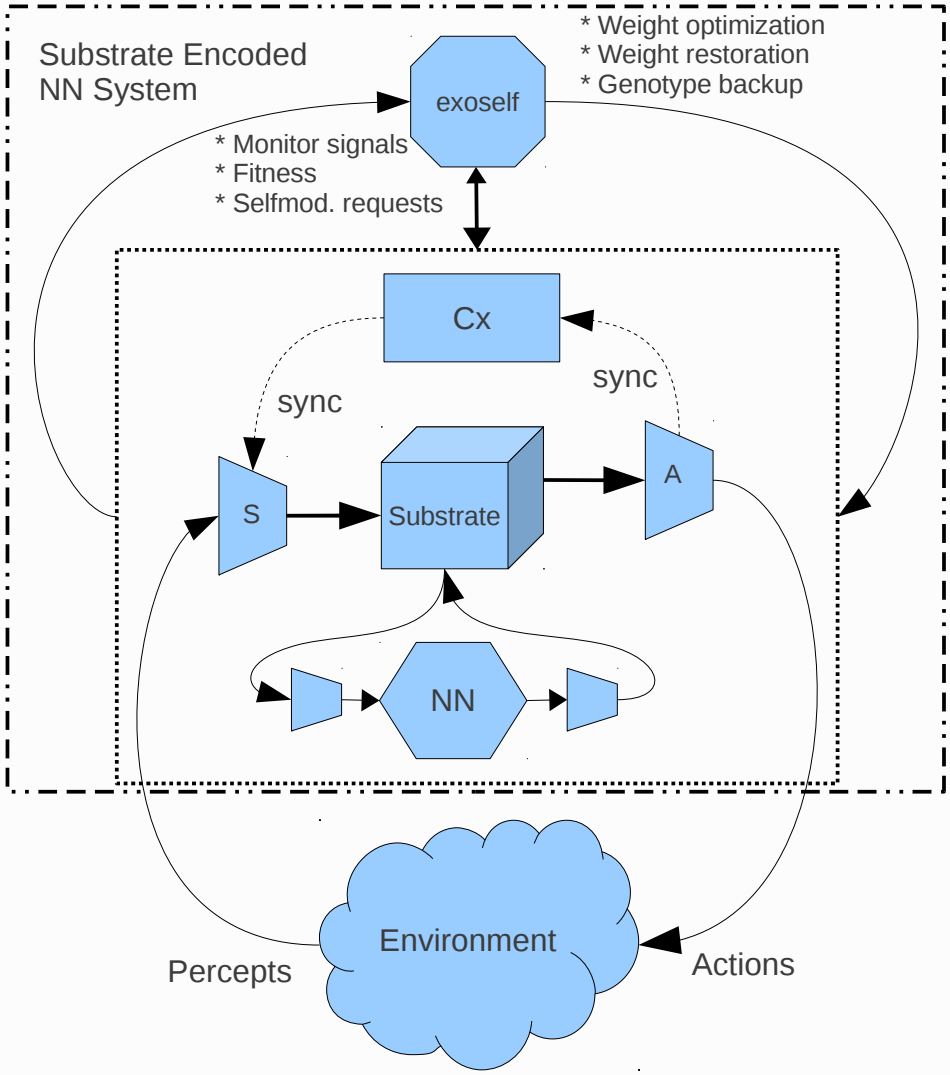
Mnesia as Storage for Genotypes

- Robust and safe
- Tuple friendly
- Easy atomic mutations
  - If any part of the mutation fails, the whole mutation is just retracted automatically

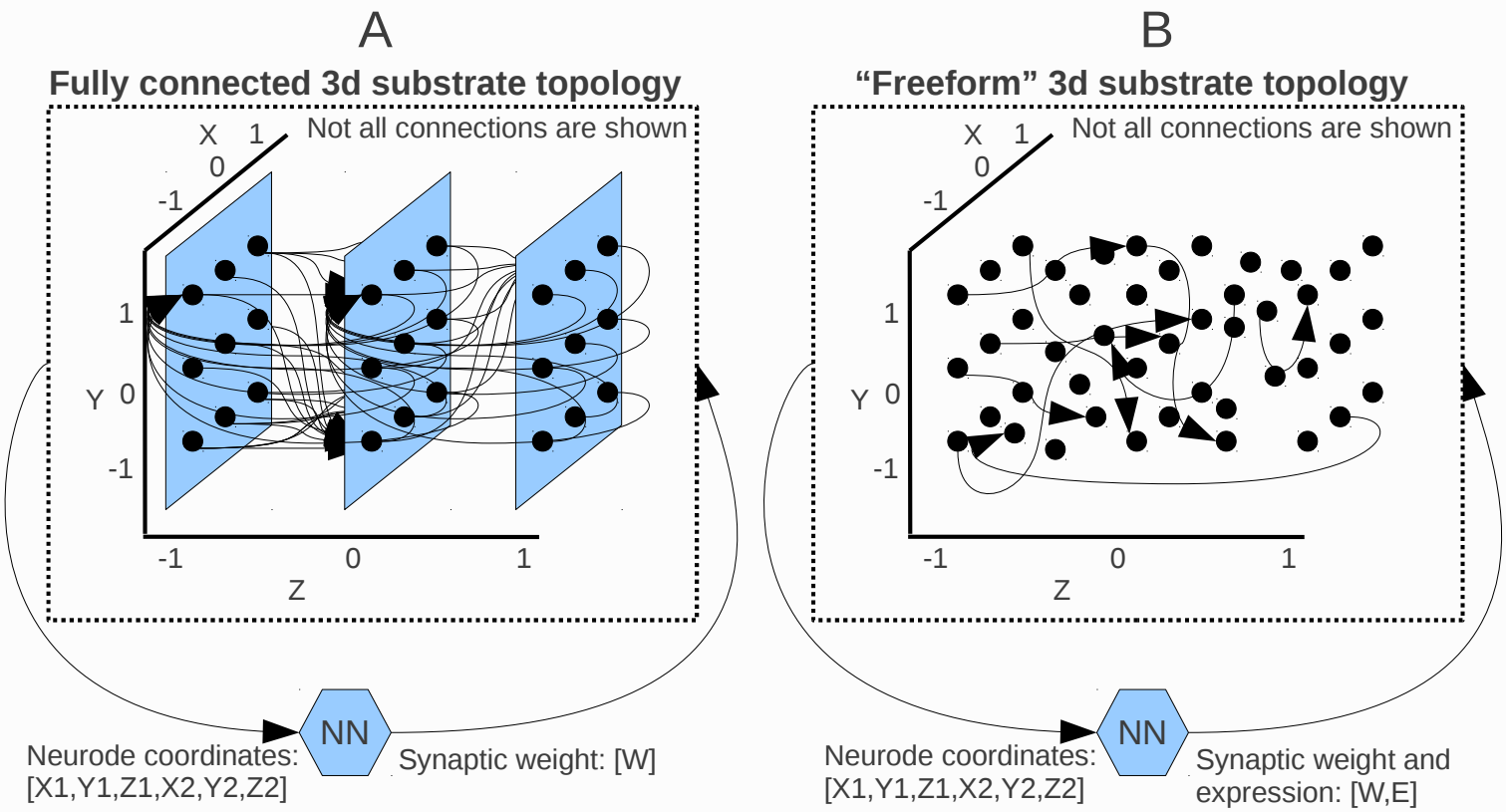
The Infomorph's Phenotype (Neural)



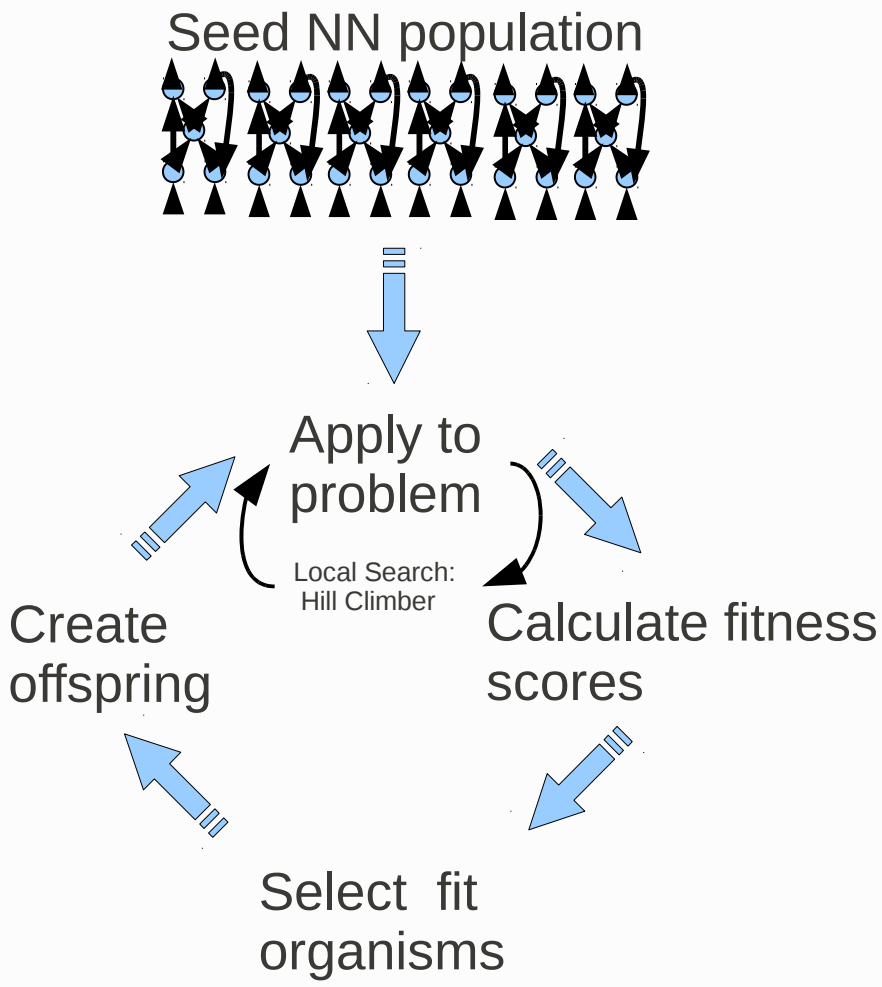
The Infomorph's Phenotype (Substrate)



Substrate Encoding (continued)

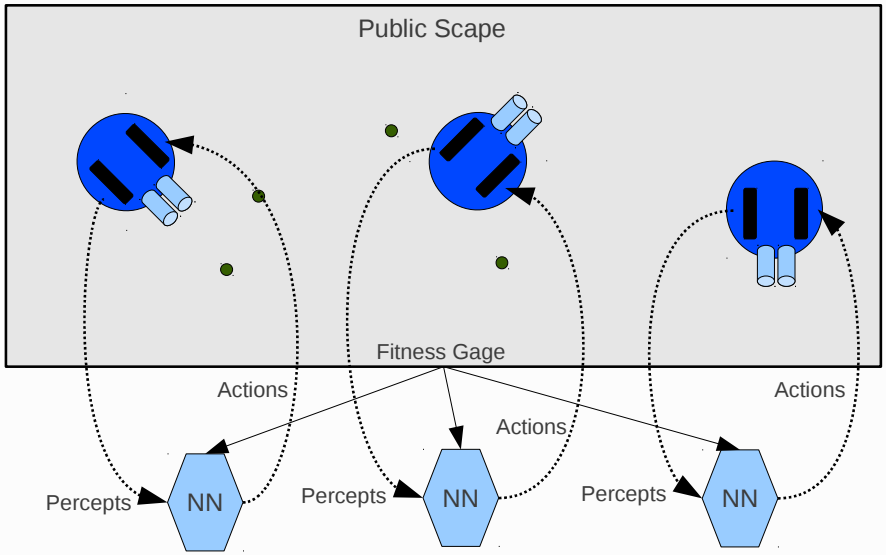


Memetic Algorithm Based TWEANN

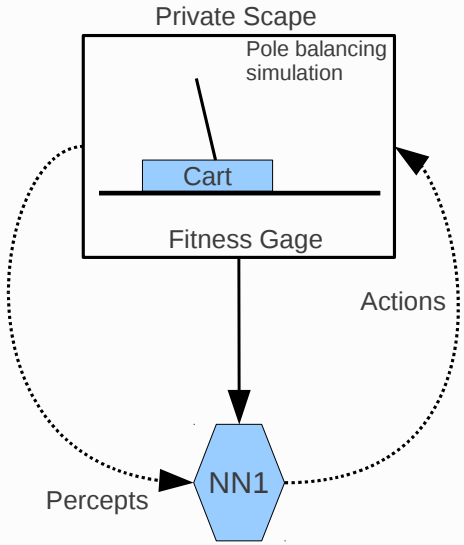


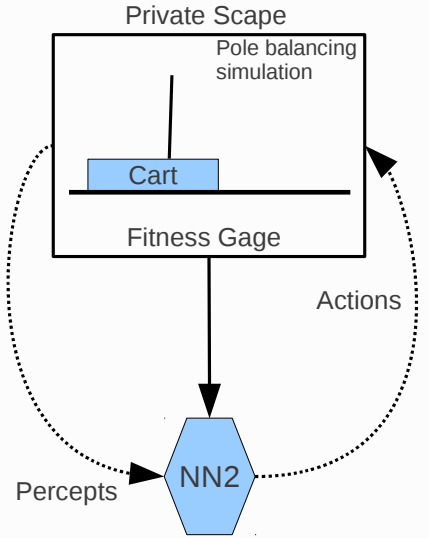
The Learning algorithm is as follows:
0. Create seed population of NN agents.
1. Spawn (convert genotype to phenotype) a population of agents.
2. Each agent interacts with the environment or some problems.
3. Each agent gets a fitness evaluation.
4. A process called exoself perturbs agent's synaptic weights.
5. Applies it to the problem again.
6. And if its performance increases, then this new synaptic weight combination is considered best, and we again perturb the synaptic weights. If the new performance is worse, then we revert to previous best, and perturb the synaptic weights.
7. Eventually all agents have had their synaptic weights tuned, and the fitness scores of the agents is compared.
8. Fitter agents allowed to create more offspring.
9. Goto: 1

Scape & Morphology

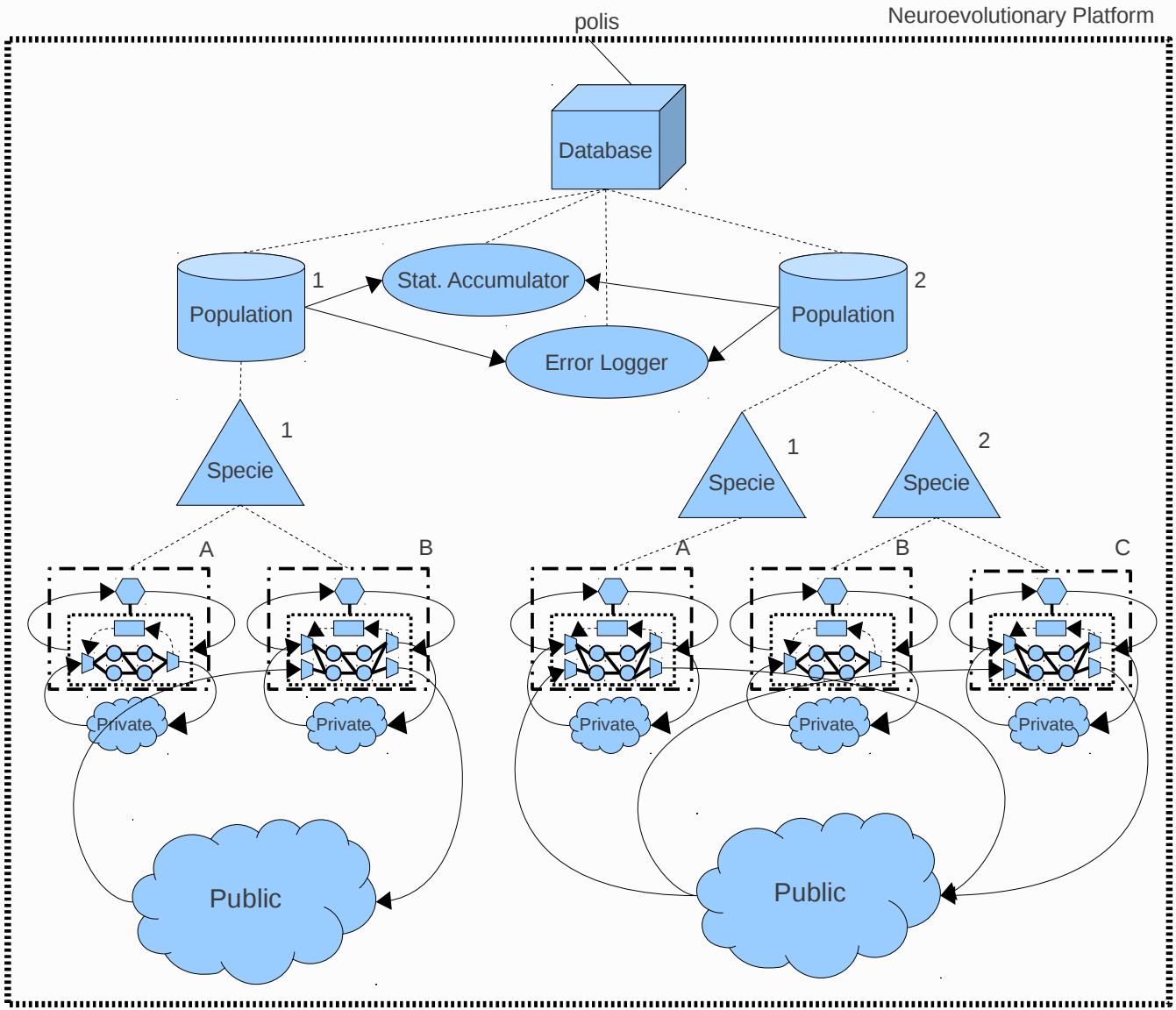


This is how my NN based agents interact with problems/simulated environments.



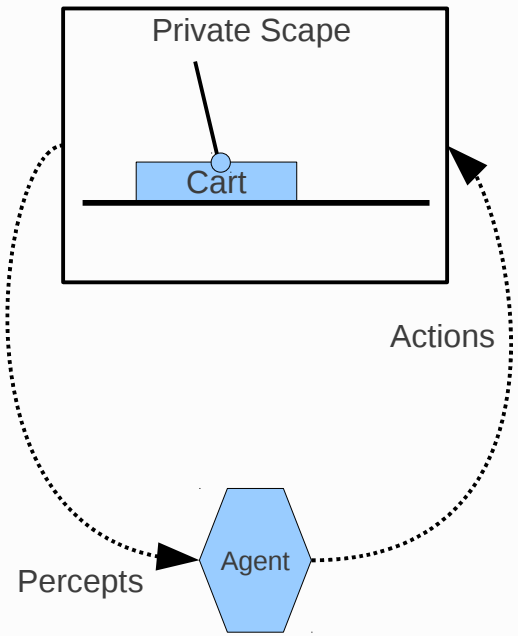


DXNN

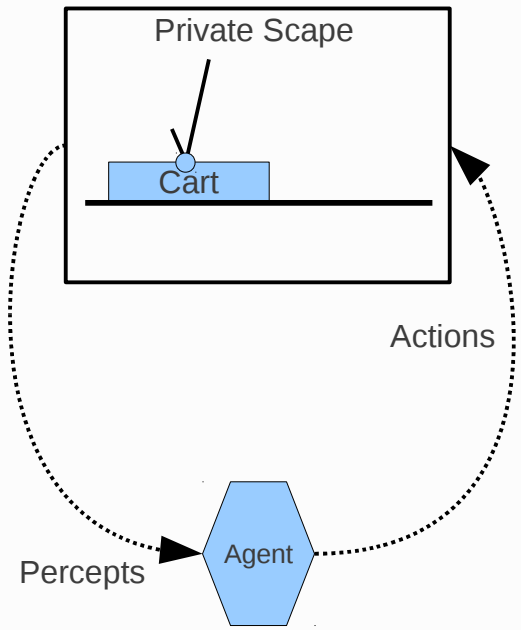

polis
Neuroevolutionary Platform
Database
Population
1
Stat. Accumulator
Population
2
Error Logger
Specie
1
Specie
1
Specie
2
A
B
A
B
C
Private
Private
Private
Private
Private
Private
Public
Public

The Pole Balancing Benchmark

A. Single pole balancing simulation



B. Double pole balancing simulation



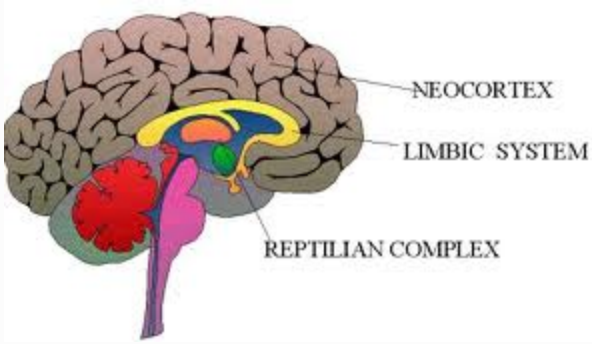
Double Pole Balancing Benchmark

Method	Without-Damping	With-Damping
RWG	415209	1232296
SANE	262700	451612
CNE	76906	87623
ESP	7374	26342
NEAT	---	6929
CMA-ES*	3521	6061
CoSyNE*	1249	3416
DXNN	***2359***	***2313***

Benchmark data taken from: Faustino “Gomez, Jurgen Schmidhuber, Risto Miikkulainen,: Accelerated Neural Evolution through Cooperatively Coevolved Synapses. Journal of Machine Learning Research 9 (2008) 937-965”

Complexification and Elaboration

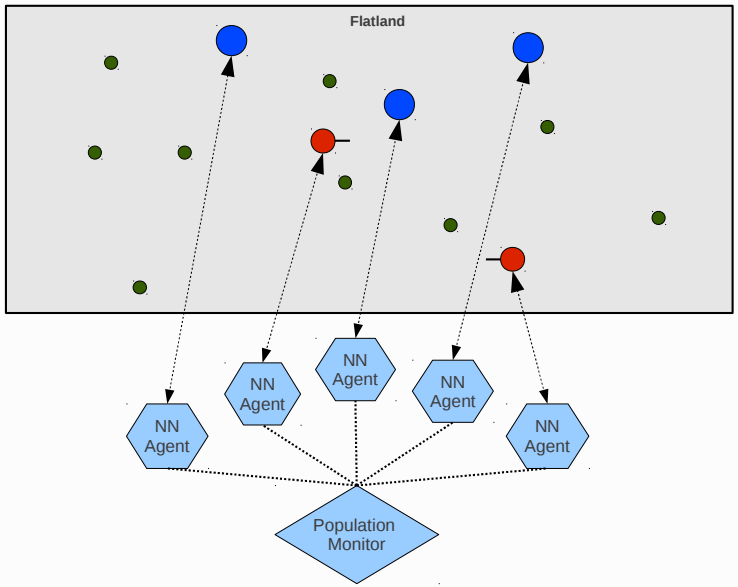
- Start with a simple initial topology
- Add to and elaborate on the topology during mutation phases
- Apply parametric mutations only to the newly created Neurons
- Scale the fitness scores based on NN size

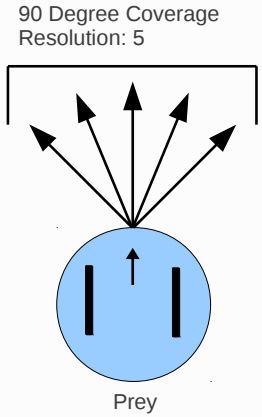


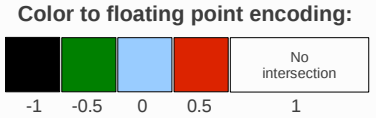
Demonstrations of Applications

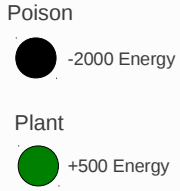
Artificial Life

- Simple Food Gathering
- Dangerous Food Gathering
- Predator Vs. Prey



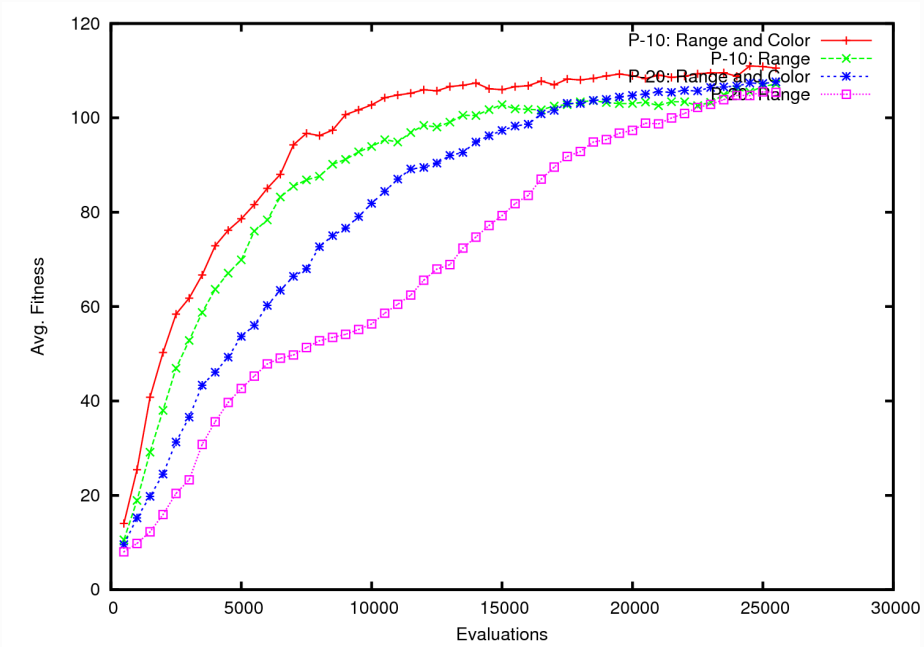


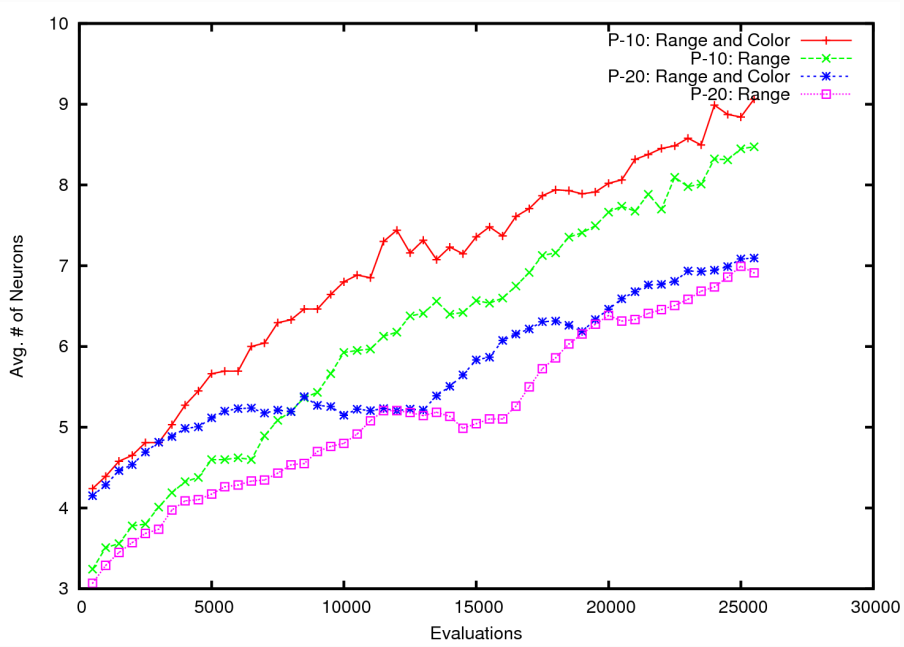




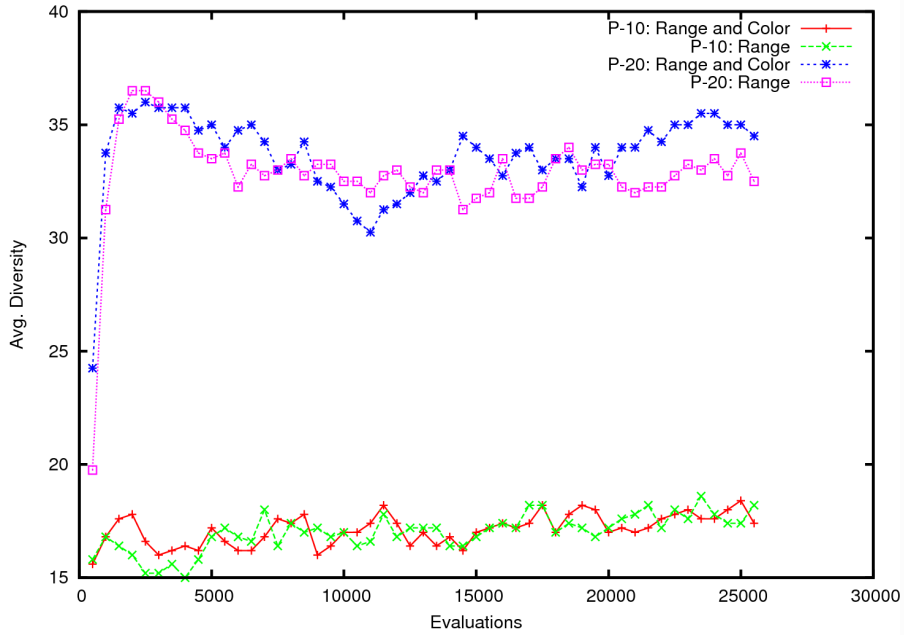
Sensors Available:
* Range Sensor:
Resolution-5
* Color Sensor:
Resolution-5

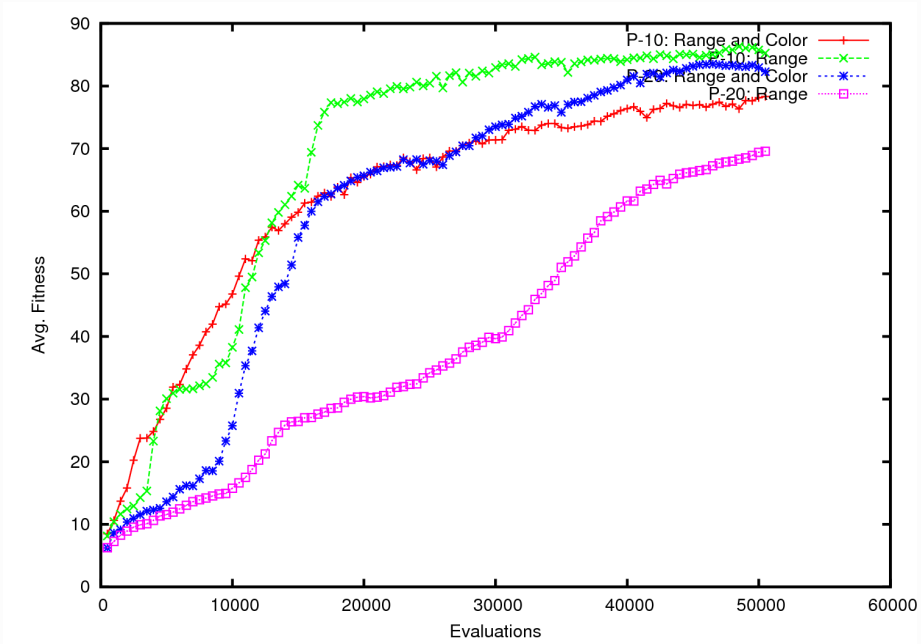
Actuators Available:
* Differential Drive

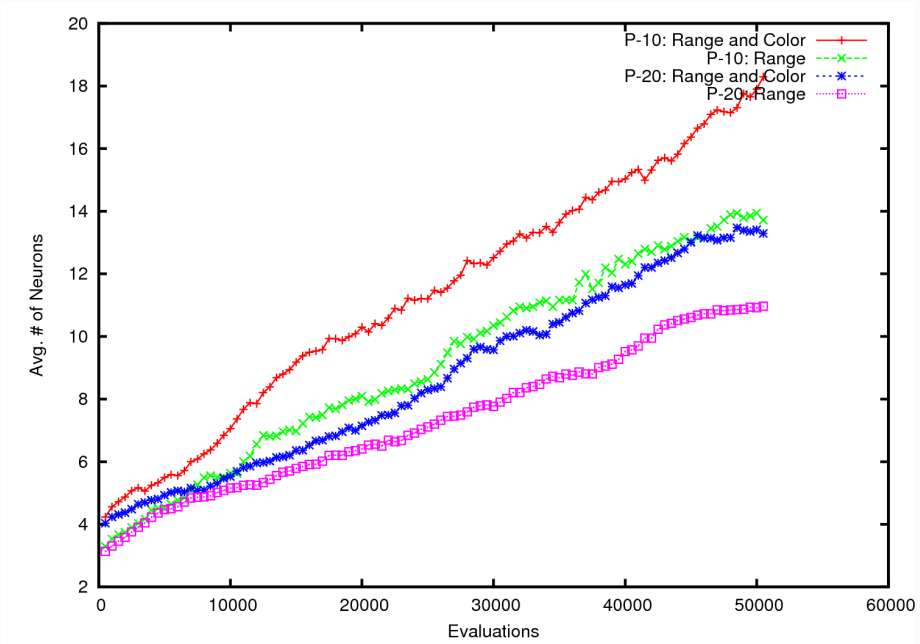




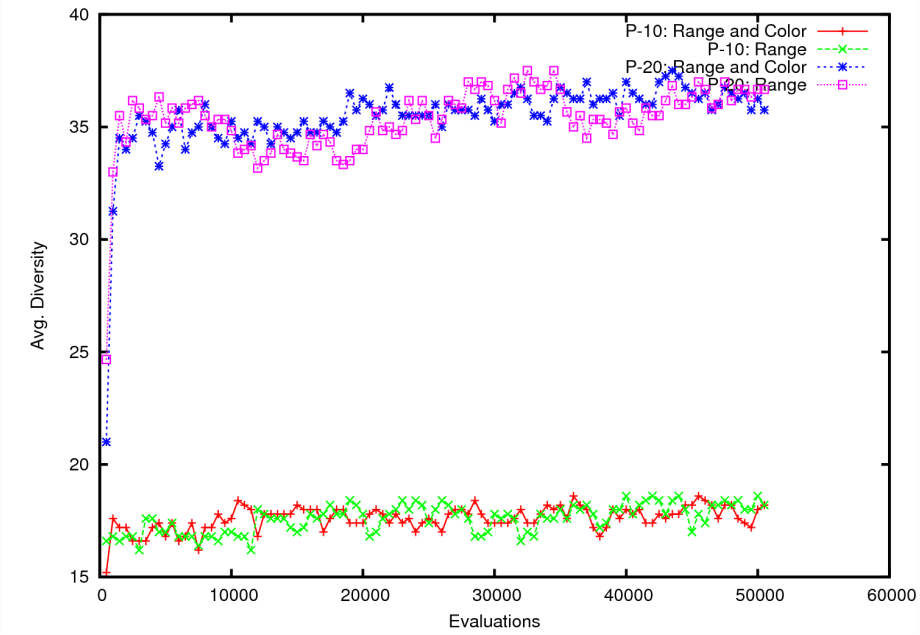
Simple Food Gathering



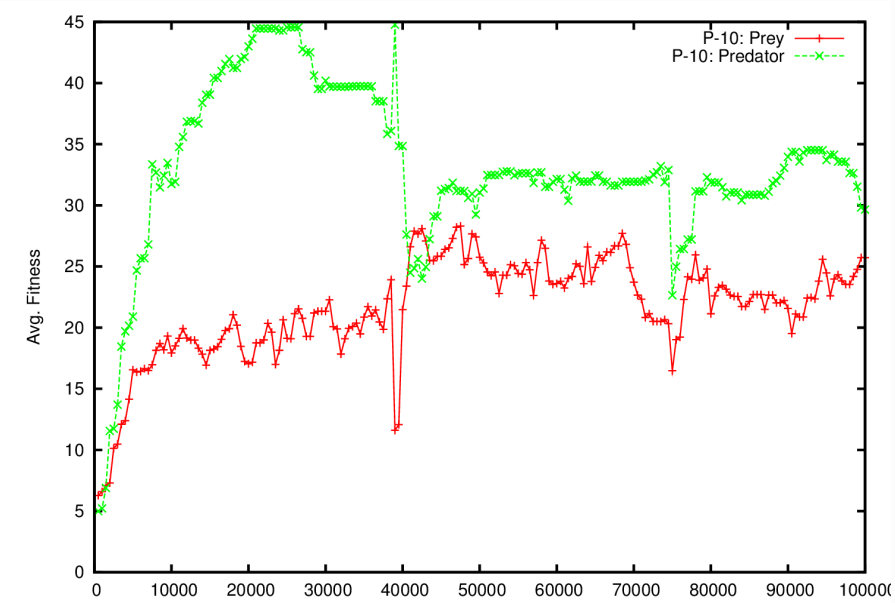


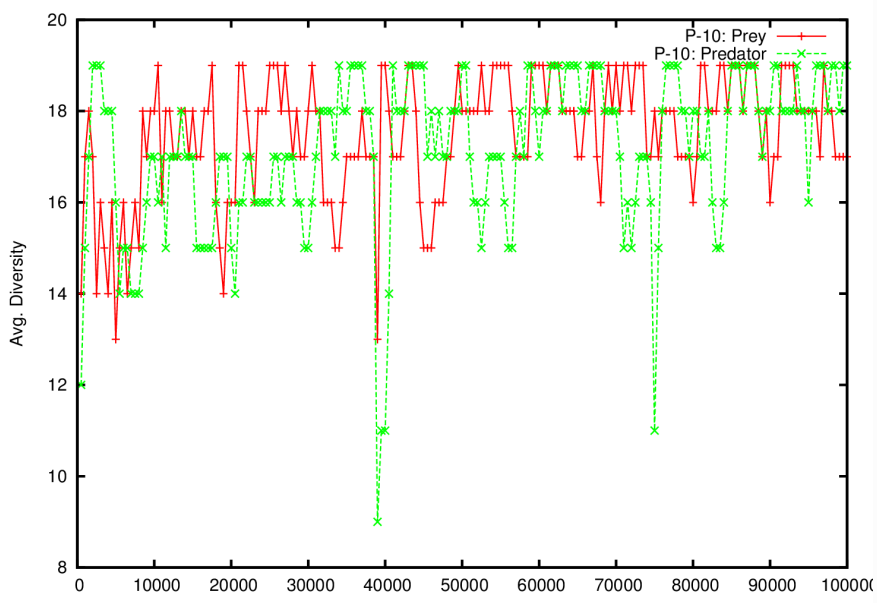


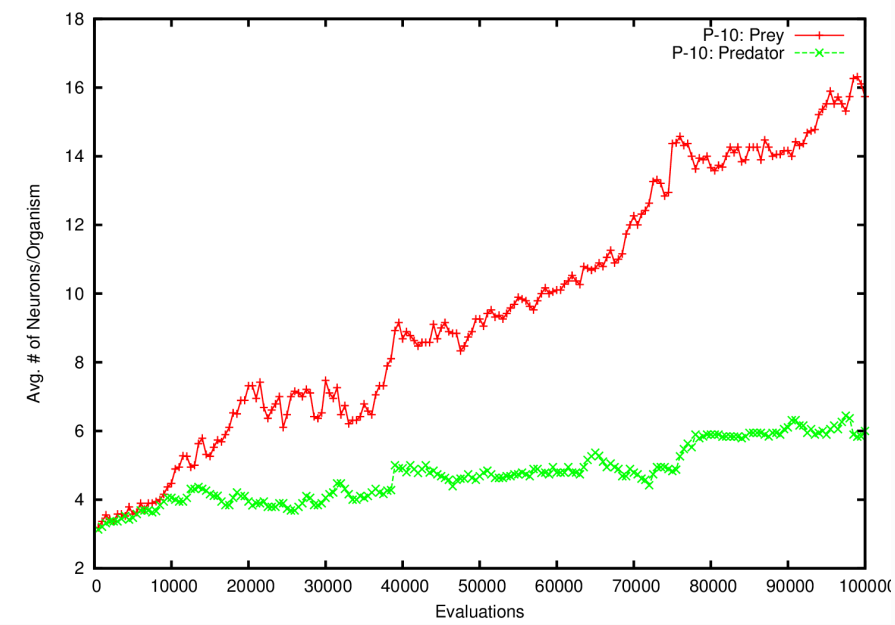
Dangerous Food Gathering

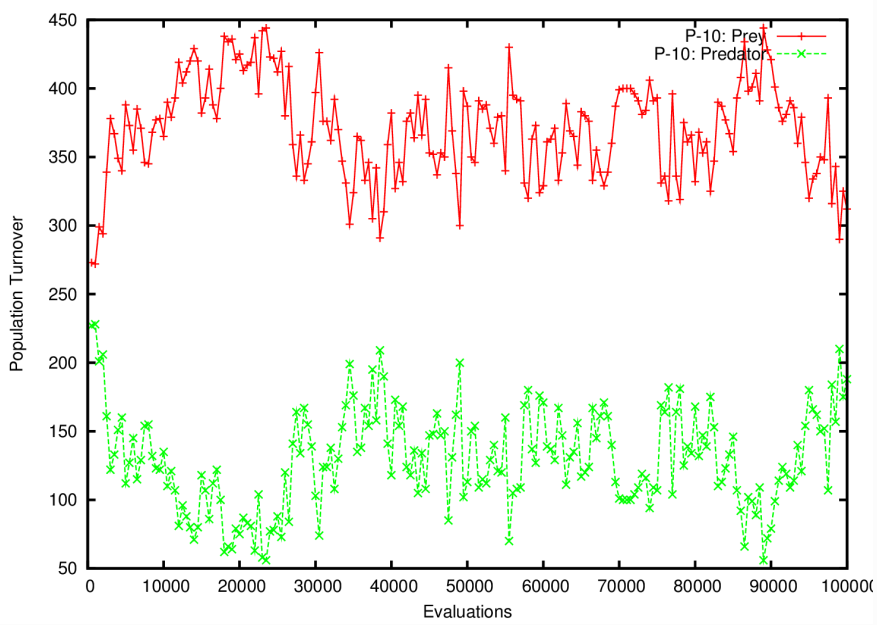


Predator Vs. Prey



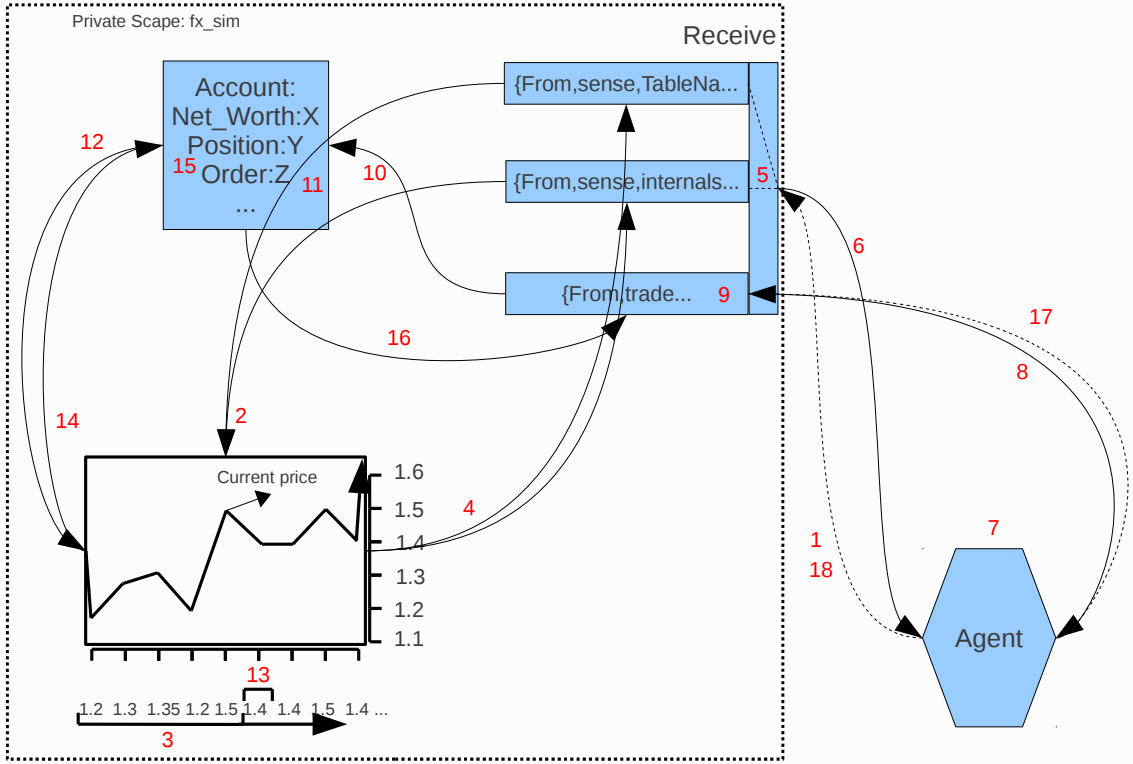




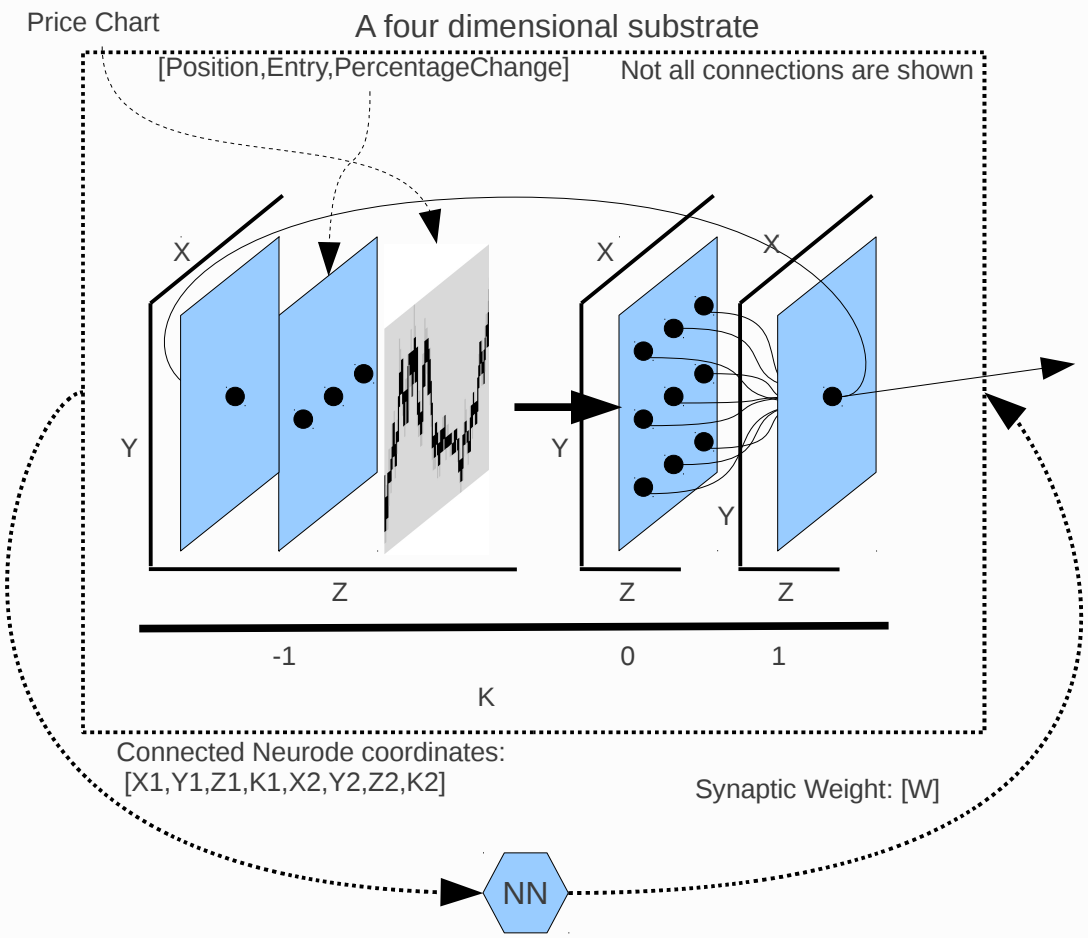


Forex Trading

- Trading using sliding window
- Trading using chart window



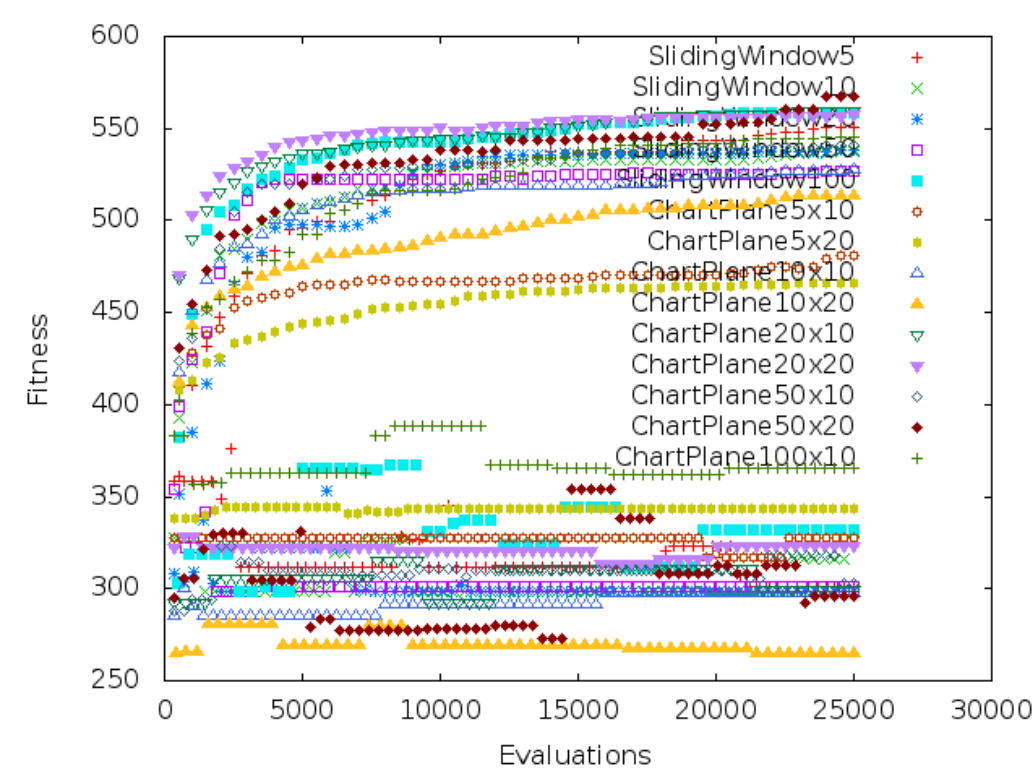
The Substrate Topology

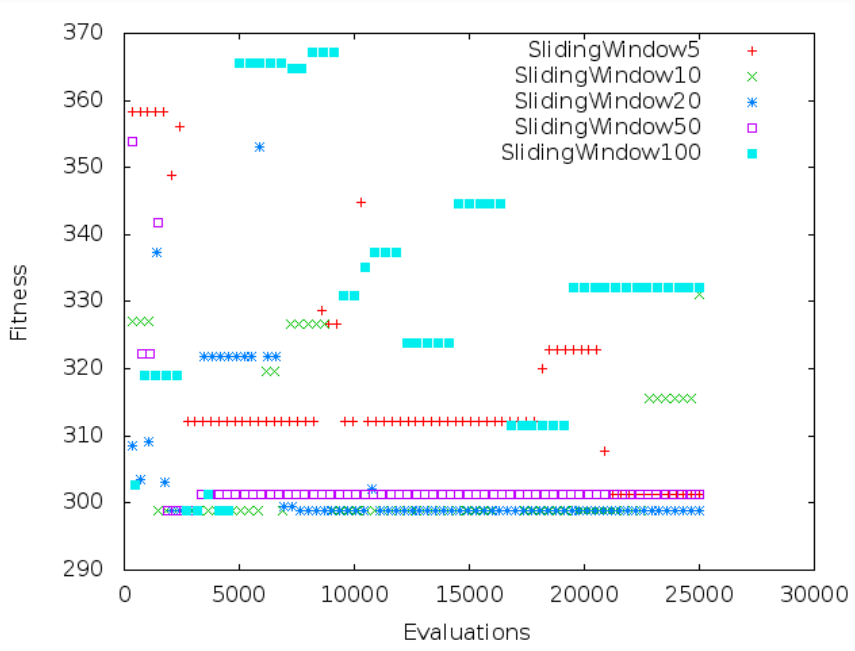


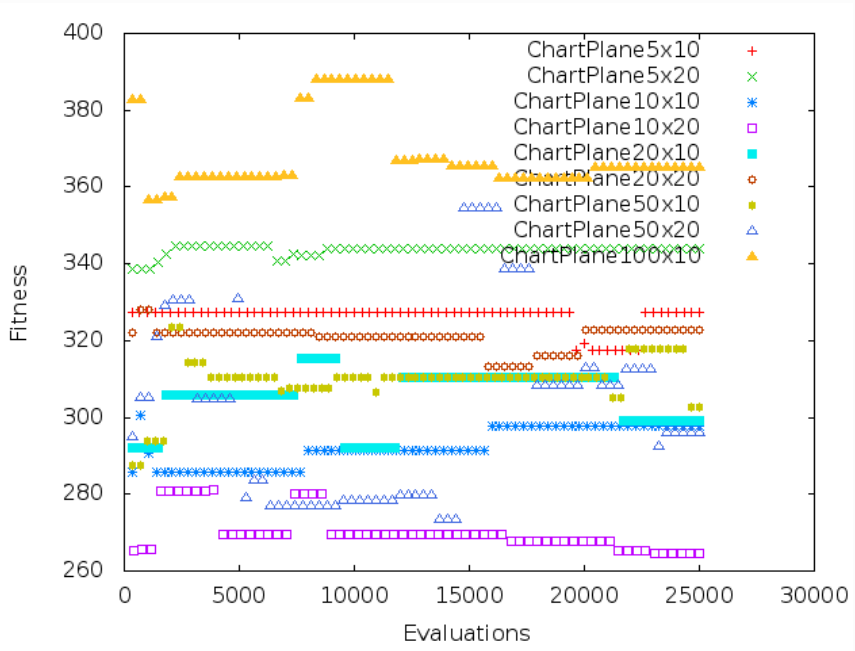
Forex Trading Results

TrnAvg	TrnBst	TstWrst	TstAvg	TstStd	TstBst	Price Vector Sensor Type
540	550	225	298	13	356	[SlidWindow5]
523	548	245	293	16	331	[SlidWindow10]
537	538	235	293	15	353	[SlidWindow20]
525	526	266	300	9	353	[SlidWindow50]
548	558	284	304	14	367	[SlidWindow100]
462	481	214	284	32	346	[ChartPlane5X10]
454	466	232	297	38	355	[ChartPlane5X20]
517	527	180	238	32	300	[ChartPlane10X10]
505	514	180	230	26	292	[ChartPlane10X20]
546	559	189	254	29	315	[ChartPlane20X10]
545	557	212	272	36	328	[ChartPlane20X20]
532	541	235	279	23	323	[ChartPlane50X10]
558	567	231	270	20	354	[ChartPlane50X20]
538	545	256	310	37	388	[ChartPlane100x10]
311	N/A	N/A	300	N/A	N/A	Buy & Hold
N/A	704	N/A	N/A	N/A	428	Max Possible

Generalization Results

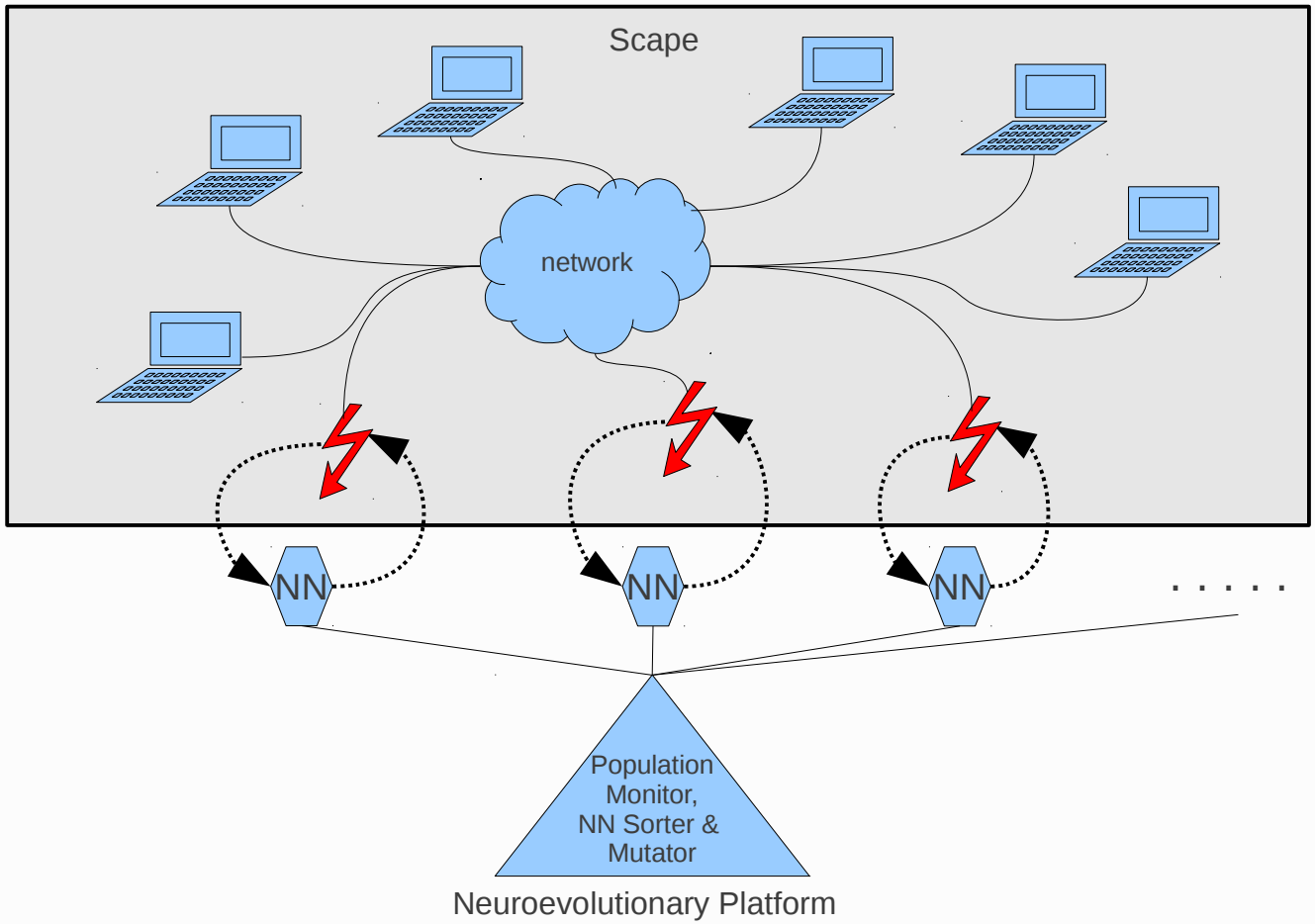




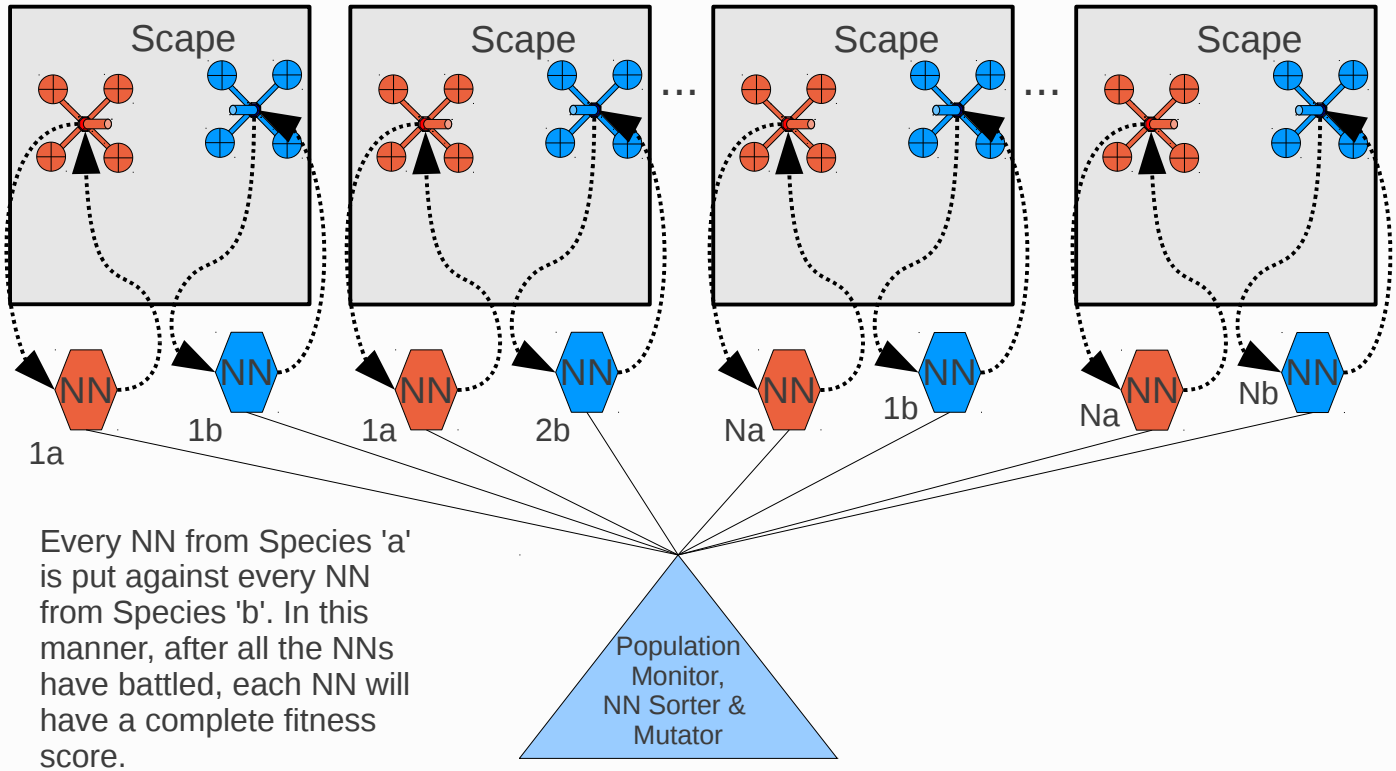


Ongoing and Future Projects

Cyberwarfare



Evolving UCAV Neurocontrollers



Evolving & Optimizing OpenSPARC

- Architecture is open and available: http://www.opensparc.net/
- Recreate the CPU architeture using DXNN's tuple balsed genotype encoding
- Evolve new modules
  - Branch predictors have been evolved for a while now.
- Optimize existing modules
- Test suite available

DXNN Research Group

- www.DXNNResearch.com
- Book: Handbook of Neuroevolution Through Erlang
  - Author: Gene Sher
  - Foreword: Joe Armstrong
  - Publisher: Springer
  - Release: Towards the end of 2012

Neural Network Research Repository

- A repository of evolved NN based agents:
  - With problem, and general setup specifications.
  - Ready to use solutions.
  - Database for general pattern mining.
- Modularized and decoupled architecture:
  - People can work on any one feature or module without interfering or worrying about the rest.
  - Add new features and modules.
  - Have access to new features and modules.
  - Crowedsourcing, rapidly expand to numerous application areas.

Thanks! Questions?

- References, Preprints, Articles...
  - Gene I. Sher (2012) ”Neuroevolution Through Erlang”, book to be published towards the end of 2012, by Springer.
  - Gene I. Sher (2012) Preprint: “Evolving Chart Pattern Sensitive Neural Network Based Forex Trading Agents”
  - Gene I. Sher (2011) Preprint: “DXNN Platform: The Shedding of Biological Inefficiencies” found at: http://arxiv.org/abs/1011.6022
  - Gene I. Sher (2011) “DXNN: Evolving Complex Organisms in Complex Environments Using a Novel TWEANN System”. Proceedings of the Genetic and Evolutionary Computation Conference 2011. Dublin, Ireland. found at:http://dl.acm.org/citation.cfm?id=2001942
  - www.DXNNResearch.com
- Get the code
  - https://github.com/CorticalComputer/
    - DXNN2 which is developed within my book, will be uploaded to github within a few weeks.
- Get in touch
  - CorticalComputer@gmail.com

END