Erlang: The Unintentional Neural Network Programming Language

Erlang Factory
San Francisco – March 2011

Gene Sher
Introduction

- The Long Standing Goals of Computational Intelligence
- Brain is just an organic substrate based NN
  - Blue Brain Project - http://bluebrain.epfl.ch/
- Computer hardware is advancing steadily
- A programming language is needed with the right features
- Erlang: As the NN Programming Language
Computational Intelligence Through Genetic Algorithms and Neural Networks
Biological Neural Network
Artificial Neural Networks

• Simulate biological NNs to various degrees of precision
• Directed graphs
• Parallel
• Learn, adapt, and generalize
An Artificial Neuron

1. Dot product:
   \[ DP = (1 \times 0.76) + (0 \times 0.46) \]
   Threshold = \((-1 \times 1)\)

2. Activation strength:
   \[ \text{Output} = \tanh(DP + \text{Threshold}) \]
The Input is Just a Vector

Output

[-0.29]

1. Dot product:
   \[ DP = (0.5 \cdot -1) + (0.2 \cdot 1) \]
   Threshold = (0 \cdot 1)

2. Activation strength:
   Output = \text{tanh}(DP + \text{Threshold})

AF: \text{tanh}
Weights: [0.5, 0.2]

[-1,1]

Input
A Neural Network

Wind Speed Later

\( \tanh(\tanh(C \cdot W) + \tanh(WSP \cdot W)) \)

\([-0.94]\)

[1, 0, -1]

\( \tanh(1^* - 0.76 + 0^* 0.46 + -1^* 1) \)

\([0.46]\)

Bias

\([-0.76]\)

\( \tanh(0.5^* 2 + 1^* -2) \)

[0.5, 1]

\( \tanh(1^* 0.5) \)

[1]

Coordinates

\([2, -2]\)

[0.5] Wind Speed Now
NN Learning & Plasticity Algorithms

• Supervised
  • Backpropagation
  • ...

• Unsupervised
  • Kohonen (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
• ....
Evolutionary Computation

- Based on evolutionary principles
- Stochastic search with a purpose
  - Survival of the fittest
- Genotype to Phenotype
- Mutation and crossover
George E. P. Box

• Improving productivity in a chemical process plant at Imperial Chemical Industries Ltd.
  1. Vary a setting
  2. See what happens
  3. ???
  4. Profit!!!
Evolutionary Computation Flowchart

1. Initialize the population
2. Create offspring through random variation
3. Evaluate fitness of each candidate solution
4. Apply selection algorithm
5. Terminate?
Simple Genetic Algorithm Example

Simple Mutations

Genotypes | Phenotypes
A 1001 | ![Phenotype](image)
B 0000 | ![Phenotype](image)
C 1010 | ![Phenotype](image)
D 0101 | ![Phenotype](image)

Gen-1

Genotypes | Phenotypes
1110 | ![Phenotype](image)
1100 | ![Phenotype](image)
1010 | ![Phenotype](image)
0101 | ![Phenotype](image)

Gen-2

Genotypes | Phenotypes
1110 | ![Phenotype](image)
1111 | ![Phenotype](image)
1010 | ![Phenotype](image)
0010 | ![Phenotype](image)

Gen-3

Crossover

Genotypes | Phenotypes
A 1001 | ![Phenotype](image)
B 0000 | ![Phenotype](image)
C 1010 | ![Phenotype](image)
D 0101 | ![Phenotype](image)

Gen-1

Genotypes | Phenotypes
1011 | ![Phenotype](image)
1101 | ![Phenotype](image)
1010 | ![Phenotype](image)
0101 | ![Phenotype](image)

Gen-2

Genotypes | Phenotypes
1011 | ![Phenotype](image)
1101 | ![Phenotype](image)
1001 | ![Phenotype](image)
1111 | ![Phenotype](image)

Gen-3

Mutation creates variation
Unfavorable mutations selected against
Reproduction and mutation occur
Favorable mutations more likely to survive
... and reproduce
Simple Hill Climber

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

2. Perturbation power!!!
   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. Perturbation power!!!
   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. Perturbation power!!!
   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.
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)
Topology and Weight Evolving Artificial Neural Networks

- Populations and Fitness Functions
- Parametric mutation operators
- Topological mutation operators
- Other mutation operators
  - Learning Algorithms, Activation Functions
  - Submodules
TWEANN Cycle

Seed NN population

Apply to problem

Create offspring

Calculate fitness scores

Select fit organisms
NN Programming Language
Necessary Features for a NN-PL
(These will sound very familiar)

- Encapsulation
- Concurrency through Neuron primitives
- Fault detection primitives
- Location transparency
- Dynamic code upgrade
Erlang's Features

- Encapsulation primitives
- Concurrency
- Fault detection primitives
- Location transparency
- Dynamic code upgrade
Neural Networks Through Erlang
The Topological 1:1 Mapping
Monolithic NN Supervision Tree
States & Functions

• Neuron
  • \{InputP\text{I}d\text{s},OutputP\text{I}d\text{s},W\text{s},A\text{F},P\text{F}\}
  • \{U\_W\text{s},O\text{utput}\}\text{ = update\_weights(Input,W\text{s},A\text{F},P\text{F})}
  • fanout(OutputP\text{I}d\text{s},Output)

• Sensor
  • \{Function,OutputP\text{I}d\text{s},Parameters\}
  • Sensory\_Signal = sensor:Function(Parameters)
  • fanout(NP\text{I}d\text{s},Sensory\_Signal)

• Actuator
  • \{Function,InputP\text{I}d\text{s},Parameters\}
  • actuator:Function(Actuator\_Signal,Parameters)
Genotype Storing and Encoding

- Tuple encoded
  - \{neuron, InputPlDs, OutputPlDs, Ws, AF, PF\}
- Relational database friendly
- Human readable
- Standard directed graph vertex and edge based
- Mutation operators are standard digraph operators
  - Add/remove Neuron/Vertex
  - Add/remove Synapse/Edge
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
Modular NN Topology Mapping
Modular NN Supervision Tree
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, requires no rewrite thanks to message passing
• Flexibility... in everything!
DXNN: A Case Study
Memetic Algorithm Based TWEANN

Seed NN population

Apply to problem

Local Search: Hill Climber

Create offspring

Calculate fitness scores

Select fit organisms
Evolving Topologies
Modular DXNN Architecture
Modular DXNN Supervision Tree
Seed Population

- Total first layer neurons = total # of sensors
- Total last layer neurons = output vector length
  - This is usually the sum of actuator vector lengths
- Choose AF, PF... randomly from the available lists
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
DXNN Genotype

- \{neuron,Id,InputPlIdPs,OutputPlIds,AF,PF\}
  - InputPlIdPs: \{[PlId1,W1]\...{PlIdn,Wn}\}
- \{module,Id,InputPlIds,FanOut,FanIn,OutputPlIds\}
- \{cortex,Id,Sensors,FanOut,FanIn,Actuators\}
- \{dxnn,Id,CortexId,ModuleIds,NeuronIds\}
  - \{available_AFs,[AF1...AFn]\}
  - \{available_PFs,[PF1...PFn]\}
  - \{available_SensorTypes,[S1...Sn]\}
  - \{available_ActuatorTypes,[A1...An]\}
  - \{available_MutationOperators,[MO1...MO2]\}
DXNN mutation operators

• Local search mutation operators
  • perturb_weights(Weights)
• Global search mutation operators
  • add_subcore(Genotype)/remove_subcore(Genotype)
  • add_sclink(Genotype)/remove_sclink(Genotype)
  • add_neuron(Genotype)/remove_neuron(Genotype)
  • add_nlink(Genotype)/remove_nlink(Genotype)
  • add_bias(Genotype)/remove_bias(Genotype)
  • change_af(Genotype)/change_plasticity(Genotype)
DXNN Platform
DXNN Benchmarks & Application Areas

- Double pole balancing
- Simple food gathering
- Dangerous food gathering
- Predator vs prey simulations
- Room navigation and new sensor acquisition
- Circuit design
- Time series analysis
- ...
Double Pole Balancing Setup
## Double Pole Benchmark


<table>
<thead>
<tr>
<th>Method</th>
<th>Evaluations</th>
</tr>
</thead>
<tbody>
<tr>
<td>RWG</td>
<td>474329</td>
</tr>
<tr>
<td>EP</td>
<td>307200</td>
</tr>
<tr>
<td>CNE</td>
<td>22100</td>
</tr>
<tr>
<td>SANE</td>
<td>12600</td>
</tr>
<tr>
<td>Q-MLP</td>
<td>10582</td>
</tr>
<tr>
<td>NEAT</td>
<td>3600</td>
</tr>
<tr>
<td>ESP</td>
<td>3800</td>
</tr>
<tr>
<td>CoSyNE</td>
<td>954</td>
</tr>
<tr>
<td>CMA-EX</td>
<td>895</td>
</tr>
<tr>
<td>DXNN</td>
<td>725</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>Evaluations</th>
</tr>
</thead>
<tbody>
<tr>
<td>RWG</td>
<td>415209</td>
</tr>
<tr>
<td>EP</td>
<td>76906</td>
</tr>
<tr>
<td>CNE</td>
<td>262700</td>
</tr>
<tr>
<td>SANE</td>
<td>7374</td>
</tr>
<tr>
<td>Q-MLP</td>
<td>1249</td>
</tr>
<tr>
<td>NEAT</td>
<td>3521</td>
</tr>
<tr>
<td>ESP</td>
<td>2359</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>Evaluations</th>
</tr>
</thead>
<tbody>
<tr>
<td>RWG</td>
<td>1232296</td>
</tr>
<tr>
<td>EP</td>
<td>87623</td>
</tr>
<tr>
<td>CNE</td>
<td>451612</td>
</tr>
<tr>
<td>SANE</td>
<td>6929</td>
</tr>
<tr>
<td>Q-MLP</td>
<td>26342</td>
</tr>
<tr>
<td>NEAT</td>
<td>3416</td>
</tr>
<tr>
<td>ESP</td>
<td>6061</td>
</tr>
<tr>
<td>DXNN</td>
<td>2313</td>
</tr>
</tbody>
</table>
Artificial Life Setup
2d Robots and Sensor Types

Sensor angle coverage, resolution=4

Prey

Poison
Plant
Predator

Ray casting based sensors
Predator vs. Prey
Evolving Hardware

AFs: [tanh, sin, abs...]

AFs: [AND, NOT, OR]
Modules as general programs

Where each Module can be externally evolved NN, a general program, a circuit, a substrate encoded NN...
QuadCopter Stabilization

Prop Speeds: [P1, P2, P3, P4]
Bipedal Gait

R Joint Angles: [R1,R2,R3]

L Joint Angles: [L1,L2,L3]
Time Series Analysis
(Through a sliding window)
Coevolution

- Environment and fitness landscape is created by the interaction of competing species
- Arms race
UAVs & Aerial Combat

Bently – Creative Evolutionary Systems, Ch-19, Discovering Novel Fighter Combat Maneuvers: Simulating Test Pilot Creativity
Cyberwarfare

(Something like metasploit, but NN provides the parameters and chooses the tools)

DETER (NetSecTestbed) can be found at http://www.isi.deterlab.net/
Conclusion & Future Work

• Memetic vs Genetic
• Spiking neural networks and fault tolerance
• Over-the-net distribution and sensing
• Artificial life, 3d systems, distributed processing...
• Towards more realistic simulators and evolving greater complexity...
Thanks! Questions?

• Learn More
  • Preprints available on arxiv.org

• Get the code
  • Will be available on github soon

• Get in touch
  • dxnn.research@gmail.com