NeurOS® and NeuroBlocks®
A Neural Operating System and Cognitive Building Blocks

Lee Scheffler, Cognitivity

Rapidly build/run portable scalable embeddable extensible cognitive systems
NeurOS and NeuroBlocks in a Nutshell

Build **cognitive systems**... 
... by linking **reusable modules** ... 
... into **directed neural graphs/circuits**

**Broad applicability**

**Rapid iterative visual flow development**

**Open, extensible, embeddable, portable, scalable, ...**
Me

• Software architect/CTO career
  – Multics, security (MIT, Honeywell)
  – networking, office automation, workstations (Prime)
  – data access (Constellation - startup)
  – data integration - **DataStage**
    (VMark/Ardent/Informix/Ascential/IBM)
  – cognitive systems (Cognitivity)

• **Maxims:** Be Useful, Make Stuff Work
Positioning

NeurOS
NeuroBlocks (functional)

"classical" AI (symbolic)
probabilistic, connectionist models

synapse simulations
spiking simulations

abstract

biological fidelity
Architecture Layers

- Cognitive system architecture
- Neural processing architecture
  - biologically inspired non-von Neumann non-procedural dataflow system
- Implementation architecture
  - conventional computers, networks
Cognitive System Challenge

sensation
perception
recognition
imagination
abstraction
prediction
memory
decision
learning
behavior
action
external world
user, environment
Inspirations/Influences

• Modular component systems
  – Erector set, MIDI, Legos, ...
• Circuit design
• Shell pipelines
• Spreadsheets
• Dataflow systems
• Braitenberg: "Vehicles"
• Signal processing
• Analog computing
• Biological brains
  – Jeff Hawkins: "On Intelligence"
Example: What's That Tune?

NeurOS Designer IDE

Drag & drop neural graph design canvas

NeuroBlocks parts bin

Output module displays

© Copyright 2017, Lee Scheffler patent pending
Basics

neural graph:

module

--events--->

module

links
neural graph:

<table>
<thead>
<tr>
<th>module</th>
<th>Group/layer of neurons or dendritic branches with similar function; state</th>
</tr>
</thead>
<tbody>
<tr>
<td>link</td>
<td>Multiplexed event signal path: axons of multiple neurons</td>
</tr>
<tr>
<td>neural graph</td>
<td>Directed flow, loops, nestable sub-graphs</td>
</tr>
<tr>
<td>event</td>
<td>New spiking rate of a neuron</td>
</tr>
</tbody>
</table>

Basics
Basics

neural graph:

<table>
<thead>
<tr>
<th>module</th>
<th>Group/layer of neurons or dendritic branches with similar function; state</th>
</tr>
</thead>
<tbody>
<tr>
<td>link</td>
<td>Mulitplexed event signal path: axons of multiple neurons</td>
</tr>
<tr>
<td>neural graph</td>
<td>Directed flow, loops, nestable sub-graphs</td>
</tr>
<tr>
<td>event</td>
<td>New spiking rate of a neuron</td>
</tr>
</tbody>
</table>

© Copyright 2017, Lee Scheffler patent pending
### Basics

**neural graph:**

<table>
<thead>
<tr>
<th>module</th>
<th>Group/layer of neurons or dendritic branches with similar function; state</th>
</tr>
</thead>
<tbody>
<tr>
<td>link</td>
<td>Multiplexed event signal path: axons of multiple neurons</td>
</tr>
<tr>
<td>neural graph</td>
<td>Directed flow, loops, nestable sub-graphs</td>
</tr>
<tr>
<td>event</td>
<td>New spiking rate of a neuron</td>
</tr>
</tbody>
</table>

© Copyright 2017, Lee Scheffler  
patent pending
Basics

neural graph:

- module: Group/layer of neurons or dendritic branches with similar function; state
- link: Multiplexed event signal path: axons of multiple neurons
- neural graph: Directed flow, loops, nestable sub-graphs
- event: New spiking rate of a neuron
neural graph:

- events - →

links

event code:

(0, id, 1.0)  (25, id, 0)  (50, id, 0.5)  (75, id, 0.33)  (120, id, 0)
## Module Types

<table>
<thead>
<tr>
<th>inputs (senses)</th>
<th>keyboard</th>
<th>temporal pattern in</th>
<th>grid in</th>
<th>data gen</th>
<th>stream in</th>
<th>MIDI in</th>
</tr>
</thead>
<tbody>
<tr>
<td>processing</td>
<td>filter</td>
<td>transformer</td>
<td>group ops</td>
<td>wrapper</td>
<td>subgraph</td>
<td></td>
</tr>
<tr>
<td>memory</td>
<td>working memory</td>
<td>set patterns</td>
<td>sequence patterns</td>
<td>temporal patterns</td>
<td>pattern reify</td>
<td></td>
</tr>
<tr>
<td>outputs (actions, effectors)</td>
<td>print</td>
<td>plot</td>
<td>cloud</td>
<td>grid out</td>
<td>stream out</td>
<td>log</td>
</tr>
</tbody>
</table>

© Copyright 2017, Lee Scheffler
patent pending
# Memory Pattern Module Types

<table>
<thead>
<tr>
<th>Sets</th>
<th>Concurrent feature collections in any order</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Semantic range:</td>
</tr>
<tr>
<td></td>
<td>any/OR, a few, some, many, most, all/AND</td>
</tr>
<tr>
<td>Sequences</td>
<td>Time-independent sequences of features</td>
</tr>
<tr>
<td></td>
<td>Parameters for non-exact sequence matching</td>
</tr>
<tr>
<td>Temporal Patterns</td>
<td>Time-relative sequences of multiple features</td>
</tr>
<tr>
<td></td>
<td>Parameters for non-exact matching, speed range</td>
</tr>
<tr>
<td>Reify</td>
<td>Inverse: generate pattern features</td>
</tr>
<tr>
<td></td>
<td>e.g., prediction, imagination, expectations, feedback</td>
</tr>
</tbody>
</table>
Layers of Patterns

F = \{a, e, f, g1\}

<table>
<thead>
<tr>
<th>letter</th>
<th>A</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>1.0</td>
<td>1.0</td>
<td></td>
<td></td>
<td>1.0</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>1.0</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>f</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>e</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>g2</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>g1</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>d</td>
<td></td>
<td></td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>i</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>l</td>
<td></td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>h</td>
<td></td>
<td></td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>m</td>
<td></td>
<td></td>
<td></td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>k</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

© Copyright 2016 Lee Scheffler
Layers of Patterns
Pattern Learning

• Current inputs match an existing pattern well-enough?
  – novelty threshold parameter
  – **adjust feature weights** of best matching pattern(s); learning rate parameter, annealing

• Otherwise:
  – create a new pattern from the current inputs

Enables complex pattern heterarchies
  – specific exemplars, stereotypes, ...
  – layers of recombination of abstractions
Typical Cognitive System Structure

- sensory preprocessing
- feature/invariant extraction
- dimensionality reduction
- multiple channels

- predictions
- expectations
- imagination

- pattern learning and recognition
- abstraction
- reification

- sensory control

- external world
  user, environment

- outputs, actions
  multiple channels
NeurOS Strengths

• One/few-shot learning
  – no huge training sets
• Continuous on-line learning
• Unsupervised, supervised ("teaching"), reinforcement learning
• Few configuration "hyper-parameters"
NeurOS Implementation Architecture

development tools

neural graphs

run-times

stand-alone

GPUs, custom chips

distributed

multi-process

single process

external libraries, systems
module & assembly libraries
A module can be described as:

\[
\{\text{outputs}_t, \text{state}_t\} = f(\text{params}, \text{inputs}_t, \text{state}_{t-1})
\]
Module Life-Cycle

- **create instance**
- **start(vtime)**
  - initialize external resources, internal state
  - schedule self-events if needed (e.g., input polling modules)
- **run(vtime)**
  - access input signals
    - new events (changed signal values)
    - current signal values
  - recompute internal dynamic parameters (e.g., learning rate)
  - compute internal state changes
  - update shared global data
  - compute and send output signal events
    - non-zero virtual time delay
    - schedule self-events (input modules, decays, randomness)
- **finish(vtime):** finalize/close external resources
Recursion

- Sub-circuits recaptured as reusable modules
Virtual Time

• **Always moves forward**
  – non-zero module processing vtime
    • variable processing vtimes model biology
  – enables (feedback) loops - vital!

• **Signals synchronized only where needed**
  – enables parallelism

• **Synchronized with real time at edges**
  – typical compromises: miss inputs, stumble
Master Work Queue/Loop
(single process, multi-thread implementation)

• Event send:
  – post event message (vtime, signal id, new value) on each output link
  – add run(module, vtime) to vtime-sorted work queue

• Next iteration:
  – get run(module, vtime)s for lowest vtime and run
    • can be concurrent
Performance, Scalability

• **Signal value changes only**
  – don’t run modules where there are no signal changes
  – good-enough approximations

• **Patterns indexed by events**
  – don’t evaluate patterns that don’t care about current signal changes

• **Multi-threading**

• **Multi-process, distributed execution**

• **Partitioning, load balancing**

• **Custom hardware**
Integrations, Extensions, Embeddings

• Configurable modules
  – formatted file I/O, database access, pipes/sockets, web services, etc.

• External library functions

• Wrap external library/program
  – stdin/stdout/stderr, APIs
  – no write-shared state among module instances; no synchronization guarantees

• Templated and custom modules
Active Projects

• Industrial machine vibration monitoring, alerting, predictive maintenance, diagnosis
• Adaptive doctor's user interfaces to EHR systems
• Virtual infant robot learning

• Redistributable dev kit
NeurOS and NeuroBlocks in a Nutshell

Build cognitive systems...  
... by linking reusable modules ...  
... into directed neural graphs/circuits

Broad applicability

Rapid iterative visual flow development

Open, extensible, embeddable, portable, scalable, ...
Contact

http://www.cognitivity.technology

(Sign up to be notified when NeurOS development kit is available)

lee@cognitivity.technology

Be useful
Make stuff work