

Attribute Grammar Encoding based upon a Generic Neural Markup Language

Facilitating the Design of Theoretical Neural
Network Models

Talib S. Hussain

BBN Technologies

openmap.bbn.com/~thussain

hussain@ieee.org

July 26, 2004

International Joint Conference on Neural Networks (Budapest, Hungary)

Overview

- There is a need for tools that facilitate systematic exploration of novel neural network architectures
- The Generic Neural Markup Language (GNML) may be used to specify a wide variety of structural and behavioural characteristics of neural network architectures.
- Attribute Grammar Encoding (AGE) may create GNML descriptions of a wide variety of neural networks using productions that explicitly specify both structural and behavioural properties.
- The family of architectures for a given AGE may be explored using evolutionary computation.

Motivation

- Two key issues in the field of neural networks are:
 - Poor scalability of NN models
 - Poor representations of NN models
- There is a need for techniques that enable:
 - Consistent representation of large families of current and novel NN architectures
 - Systematic development of novel NN models with improved capabilities

Background: Neural Network Simulators and Specification Languages

- NN simulation environments generally focus on parameterized specifications of neural networks
 - Drag and Drop, Library of known models
 - e.g., MATLAB Neural Network Toolbox, JavaNNS, NeuroSolutions
- NN specification languages focus on capability to program complex NNs from simple components
 - e.g., AXON, CONNECT, EpsilonNN
- Some simulators have scripting languages that enable more robust programming of components
 - e.g., PDP++, GENESIS
- Drawback: Limited to the specification of a small number of structural and behavioural properties

Background: Modular Neural Networks

- Address scalability issues
- Provide first steps towards systematic integration of NN architectures
 - Building blocks (Happel and Murre, 1992),
Cooperative modules, Redundant modules,
Mixtures of Experts (Jordan and Jacobs, 1993)
- Drawback: Limited in the heterogeneity of modules and in the variety of modular relationships

Background: Evolutionary Neural Network Systems

- Address systematic exploration of variations to a NN model
- Provide first steps towards consistent representations of broad families of NN architectures
 - Direct, Parametric, Grammatical (Jacob and Rehder, 1993), Cellular (Gruau, 1995), and Learning Rule (Bengio et al., 1994) encodings
- Drawback: Limited in complexity of NN architectures and in variety of structural and behavioural properties

AGE+GNML Approach

- Our approach to an effective NN modeling tool
 1. Use (context-sensitive) attribute grammar productions to capture neural design principles with both structural and behavioural elements (e.g., learning and processing behaviours)
 2. Use attribute grammar evaluation rules to create GNML specifications of neural networks
 3. Use generic interpreter to execute and train GNML-specified neural networks.
 4. Use (context-free) parse trees from attribute grammar as basis for a genetic search among alternatives

Background: Attribute Grammars

- Attribute grammars (Knuth, 1968) are an extension of context-free grammars (CFGs)
 1. Assignment of attribute values to the non-terminal and terminal symbols of the grammar
 2. Addition of context-sensitive attribute evaluation rules to the grammar productions.
- Attributes may be synthesized (bottom-up) or inherited (top-down).
- The value of an attribute of a symbol within a production may depend only upon the values of attributes of symbols in the same production.
- Enables hierarchical design and semantic distinction between different components.

GNML Format

- XML-based
- Specifies network as list of neurons and a list of the connections between neurons
- Allows multiple types of signals in network
- Neuron behaviours are specified explicitly within XML
 - A neuron consists of a sequence of (nested) functions and a set of memory variables of different types.
 - Functions may affect memory and outgoing signals
- Current design of generic interpreter executes each neuron once per cycle.

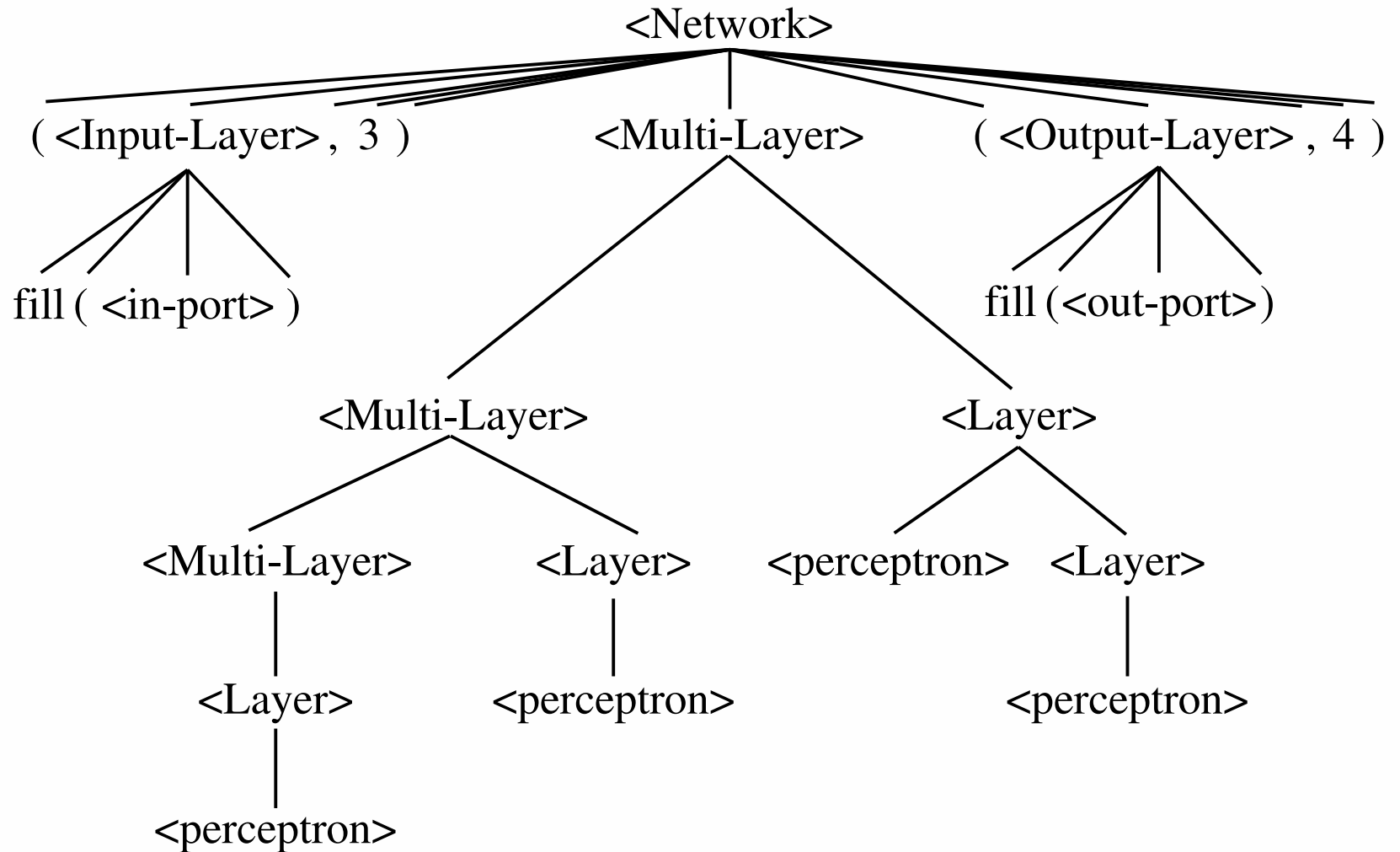
High-Level GNML Format

```
<Network>
  <InSource id='i1' signalType='Activation' />
  <InSource id='i2' signalType='Activation' />
  <InSource id='i3' signalType='Feedback' />
  <OutTarget id='o1' signalType='Activation' />
  <Neuron id='1'>
    <NeuronSpec> ... </NeuronSpec>
  </Neuron>
  <Neuron id='2'>
    <NeuronSpec> ... </NeuronSpec>
  </Neuron>
  ...
  <Connection source='i1' target='1' signalType='Activation' />
  <Connection source='1' target='2' signalType='Activation' />
  ...
  <Connection source='i3' target='2' signalType='Feedback' />
</Network>
```

Attribute Grammar Encoding

- An attribute grammar is used to represent a family of neural network architectures.
 - Extends research on the representation of NNs using CFGs (Jacob and Rehder, 1993) or CFG-similar mechanisms (Gruau, 1995).
- Grammar symbols have different roles, such as representing different structural or behavioural components at varying levels of detail.
- Symbol attributes have different purposes, such as imposing structural or behavioural constraints, and composing structural or behavioural elements.
- A complete network representation may be formed in the attributes of the root symbol.

Network Topology Example: Parse Tree



Complete GNML Specification at Root

- We may collect a complete specification of a single neural network in the attributes of the root symbol as sets of neurons and connections, from which a complete GNML specification may be readily formed.

<Network> → (<Input-Layer>, 3) <Multi-Layer> (<Output-Layer>, 4)

(synthesized)

$\langle \text{Network} \rangle . \text{all_nodes} = \langle \text{Multi-Layer} \rangle . \text{all_nodes} \cup \langle \text{Input-Layer} \rangle . \text{all_ports} \cup \langle \text{Output-Layer} \rangle . \text{all_ports}$

$\langle \text{Network} \rangle . \text{all_conns} = \langle \text{Multi-Layer} \rangle . \text{all_connections} \cup \text{full_connect} (\langle \text{Input-Layer} \rangle . \text{all_ports}, \langle \text{Multi-Layer} \rangle . \text{in_nodes}) \cup \text{direct_connect} (\langle \text{Multi-Layer} \rangle . \text{out_nodes}, \langle \text{Output-Layer} \rangle . \text{all_ports})$

$\langle \text{Network} \rangle . \text{GNML} = \text{transform} (\langle \text{Network} \rangle . \text{all_nodes}, \langle \text{Network} \rangle . \text{all_conns})$

(inherited)

$\langle \text{Input-Layer} \rangle . \text{size} = 3$

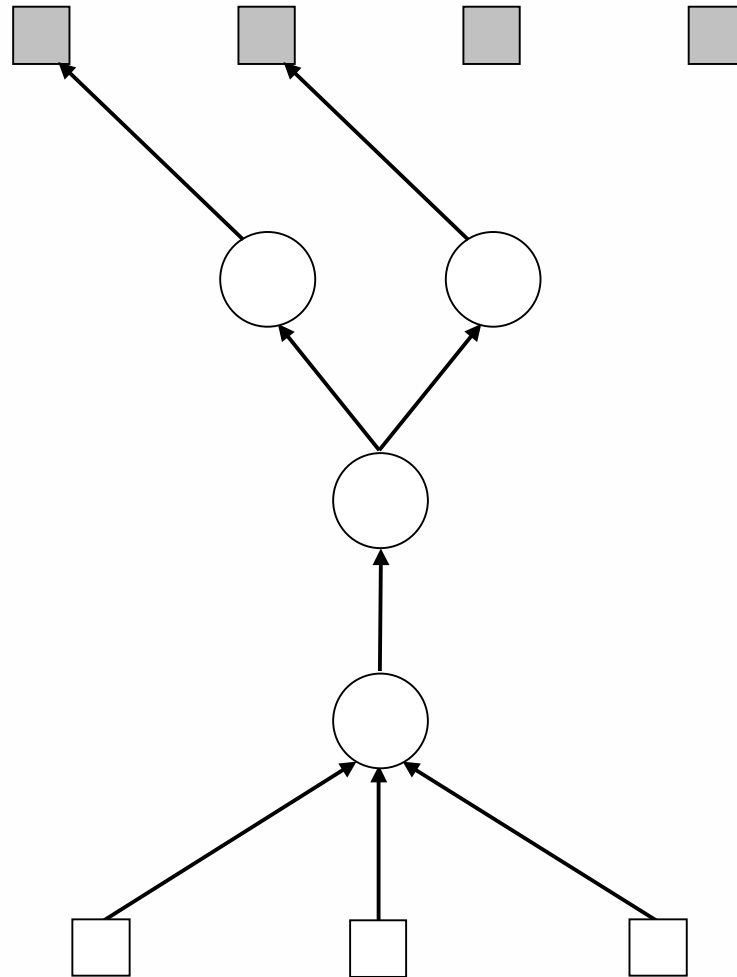
$\langle \text{Output-Layer} \rangle . \text{size} = 4$

$\langle \text{Multi-Layer} \rangle . \text{max_layers} = 3$

$\langle \text{Multi-Layer} \rangle . \text{max_size} = 3$

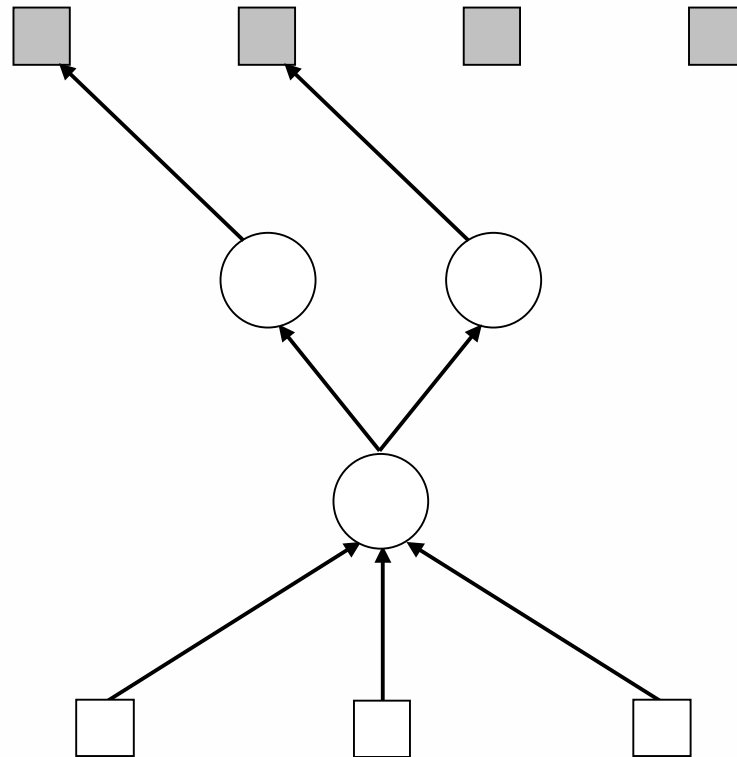
Network Topology Example: Neural Network Structure

- $max_layers = 3$



Network Topology Example: Neural Network Structure

- $max_layers = 2$



Variety of Topological Properties

- Multiple signal types
 - Connections between nodes may be explicitly associated with a specific signal type (e.g., activation or feedback)
 - Enables explicit feedforward and feedback paths
 - Enables control architectures
- Multiple Types of Module
 - Distinct module-subgrammars may be incorporated into a single compound grammar
 - Enables the specification of NNs with modules of differing architectures
 - Enables the explicit representation of different relationships between modules

Explicit Inter-Module Relationships

- Distinct topological relationships between modules may be explicitly represented in distinct productions.

<Network> → <Input-Layer> <Module> <Output-Layer>

<Module>₁ → successive-full (<Module>₂ , <Module>₃)

<Module>₁ → successive-1to1 (<Module>₂ , <Module>₃)

<Module>₁ → parallel (<Module>₂ , <Module>₃)

<Module> → <Multi-Layer>

<Module> → <Grid>

<Module> → <Recurrent-Layer>

Explicit Task Decomposition

- A module may be explicitly assigned a specific subtask of the overall problem.

<Network> → <Input-Layer> <Task> <Output-Layer>

<Task>₁ → (input <Partition>₁) <Task>₂ <Task>₃ (output <Partition>₂)

<Task>₁ → (input <Partition>) <Task>₂ <Task>₃

<Task>₁ → <Task>₂ <Task>₃ (output <Partition>)

<Task>₁ → redundant (<Task>₂ <Task>₃)

<Task> → <Module>

<Partition> → (<Subset> , complement)

<Partition> → (<Subset>₁ , complement + overlap(<Subset>₂))

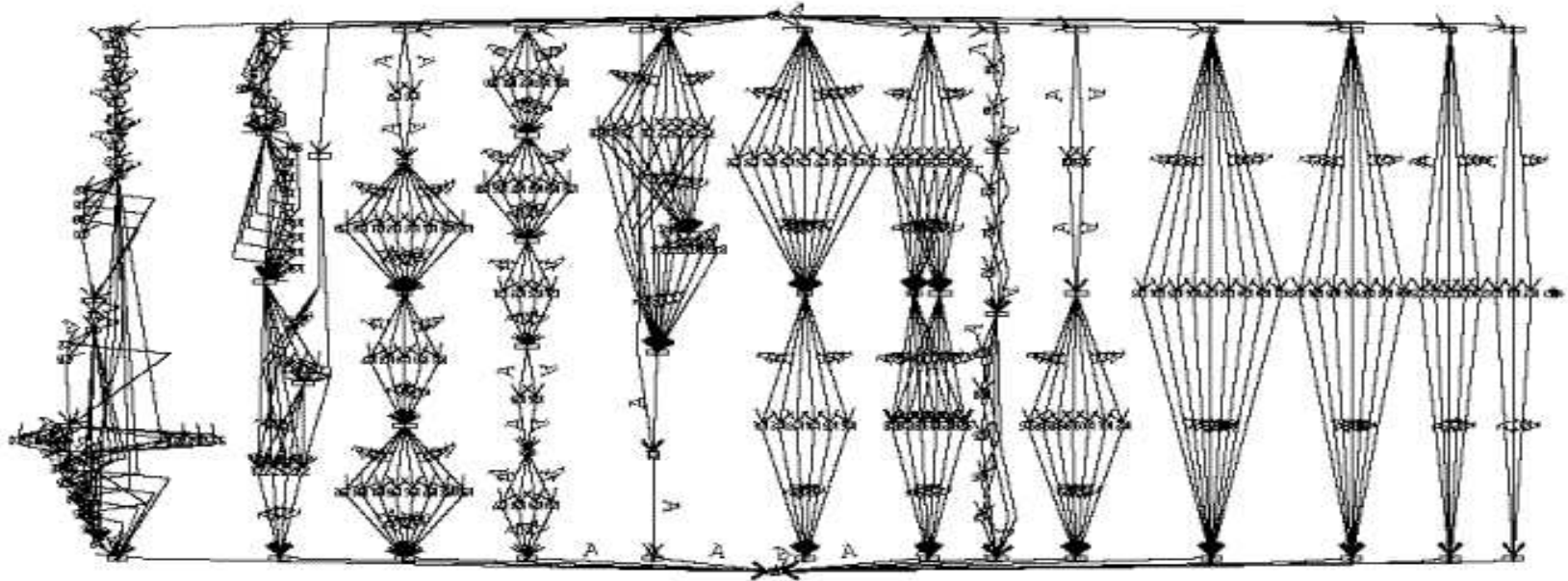
<Subset> → all

<Subset> → none

<Subset> → first-half

<Subset> → second-half

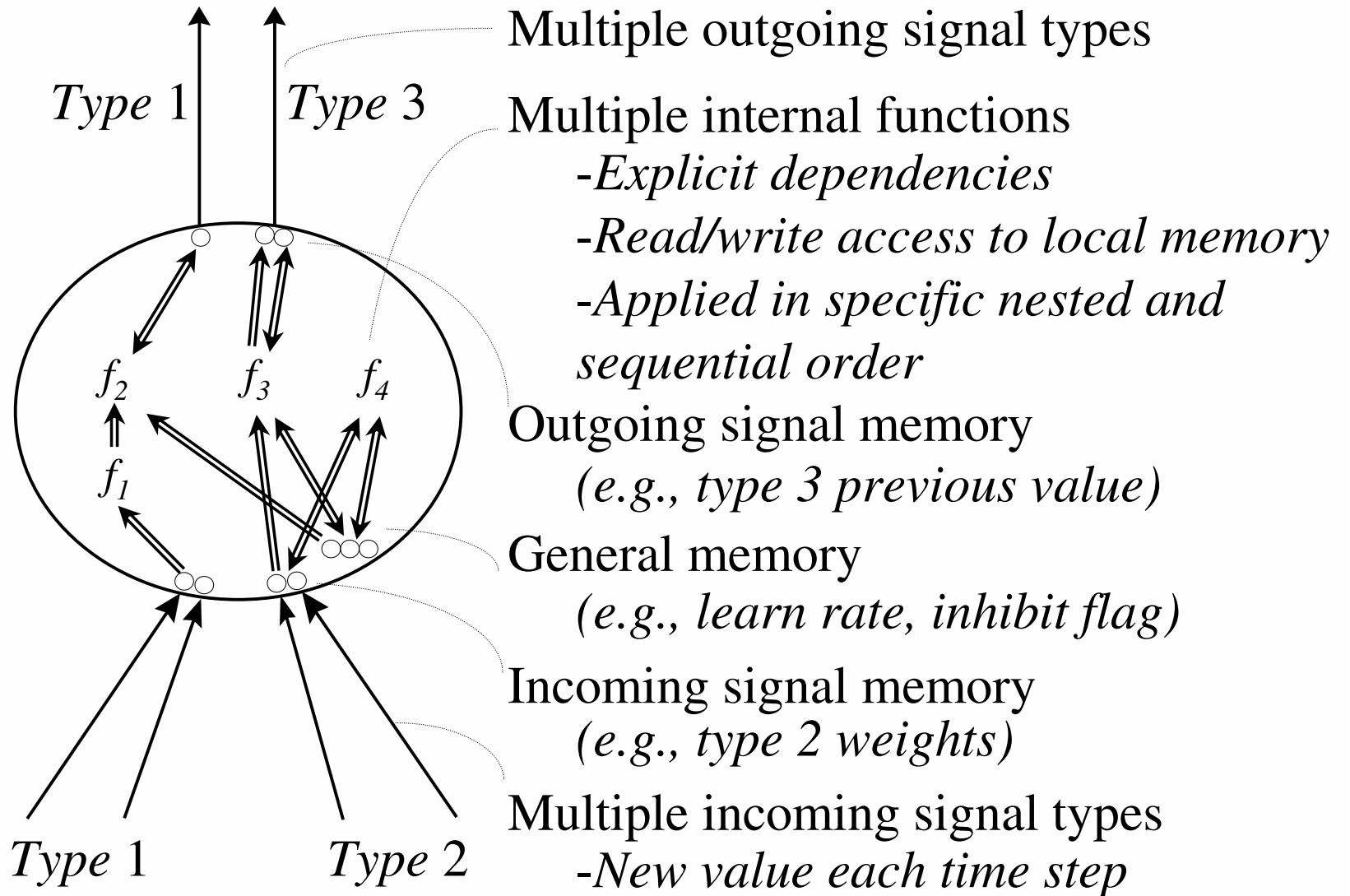
Example: Complex Modular Network



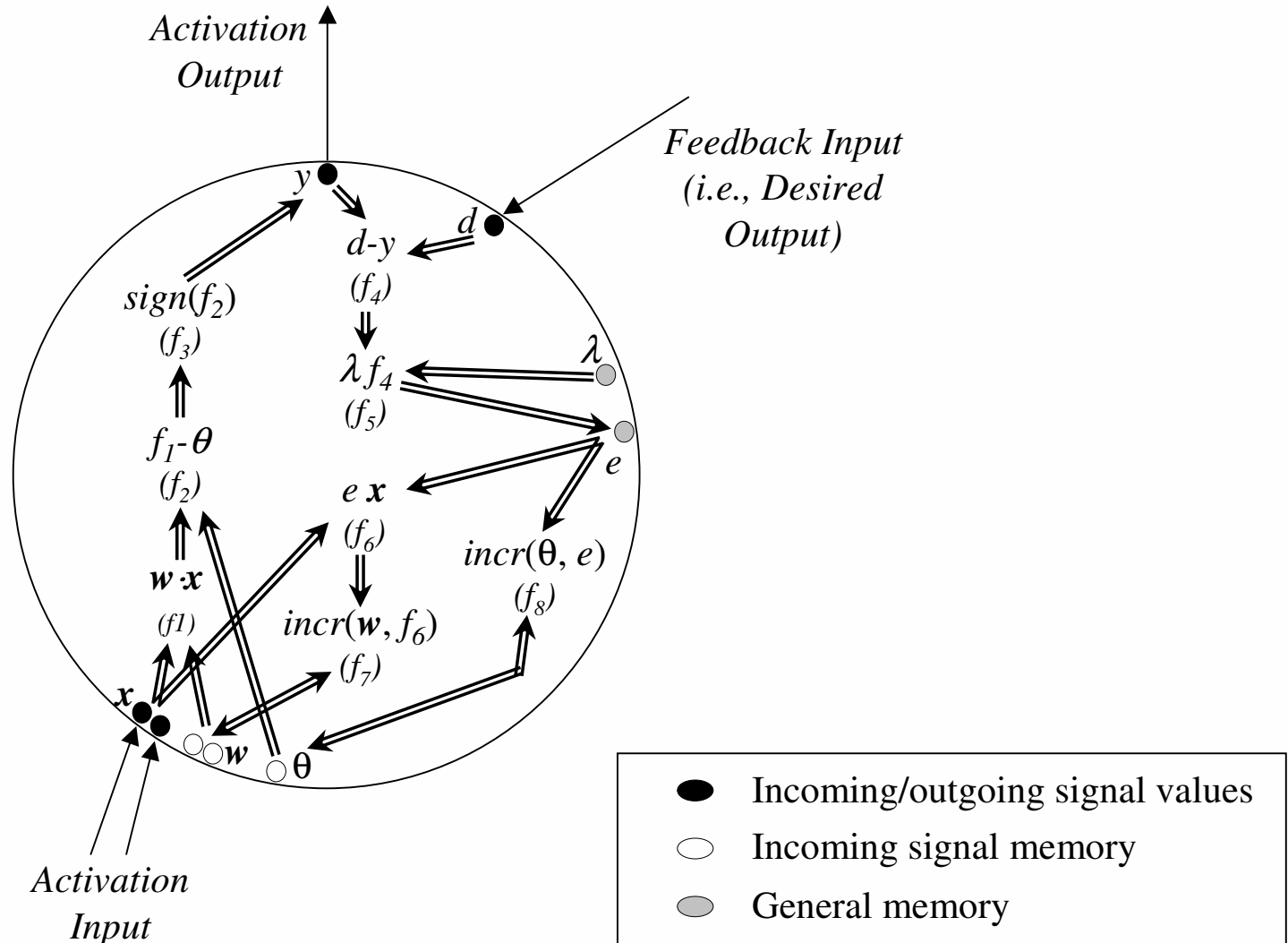
Neuron Behaviour Specification

- Focus in current research on representations of neural networks is primarily upon topology or behaviour, not both.
- GNML provides AGE with the important capability of representing neural behaviours in the same grammar as neural topologies.
- The behaviour of each neuron in the network may be specified as a distinct GNML string.
- A neuron consists of a sequence of functions and a set of memory variables of different types.
- Each neuron behaviour string is decoded by a GNML interpreter to form a functional neuron.

Enhanced AGE Neuron Model



Example: Perceptron



Neuron Behaviour Example: Context Free Base

- A neuron consists of a sequence of functions and a set of memory variables of different types.

<Neuron>	→ <enhanced-neuron> (<FunctionSequence>)
<FunctionSequence>	→ <FunctionSequence> <Function>
	→ <Function>
<Function>	→ <FunctionScalar>
	→ <FunctionVector>
<FunctionScalar>	→ <subtract> (<ReadScalar> , <ReadScalar>)
	→ <signum> (<ReadScalar>)
	→ <weighted-sum>
<FunctionVector>	→ <vector-scale> (<ReadScalar> , <WriteVector>)
<ReadScalar>	→ <FunctionScalar>
	→ <MemoryScalar>
<WriteScalar>	→ <MemoryScalar>
<ReadVector>	→ <FunctionVector>
	→ <MemoryVector>
<WriteVector>	→ <MemoryVector>

Complex Behaviour Templates

- Templates may fully or partially specify a given neural behaviour component.
- Thus, complex behaviours may be captured within a single terminal symbol as well as composed through multiple grammar productions.

`<weighted-sum>.template =`

```
"<FunctionSpec    nativeCode='dotProduct'  
                  numReadParams='2'  
                  returnType='scalar' >  
  <ReadParam dataType='vector' >  
    <LinkMemorySpec signalType='Activation'>  
      <VariableSpec name='currentValue' />  
    </LinkMemorySpec>  
  </ReadParam>  
  <ReadParam dataType='vector' >  
    <LinkMemorySpec signalType='Activation' >  
      <VariableSpec name='weight' />  
    </LinkMemorySpec>  
  </ReadParam>  
</FunctionSpec>"
```


Specification of Both Topology and Behaviour

- A complete neural network specification is formed by incorporating the GNML string specification of a neuron's behaviour into the sets that represent the topology of the network and finally transforming those sets to GNML.

<Network> → ...

$\langle \text{Network} \rangle . \text{GNML} = \text{transform} (\langle \text{Network} \rangle . \text{all_nodes}, \langle \text{Network} \rangle . \text{all_conns})$

<Layer>₁ → <Neuron> <Layer>₂

$\langle \text{Layer} \rangle_1 . \text{nodes} = \langle \text{Layer} \rangle_2 . \text{nodes} \cup$
 $[\text{concatenate}(\langle \text{Layer} \rangle_1 . \text{id}, ".1"), \langle \text{Neuron} \rangle . \text{spec}]$

<Layer> → <Neuron>

$\langle \text{Layer} \rangle . \text{nodes} = [\text{concatenate}(\langle \text{Layer} \rangle . \text{id}, ".1"), \langle \text{Neuron} \rangle . \text{spec}]$

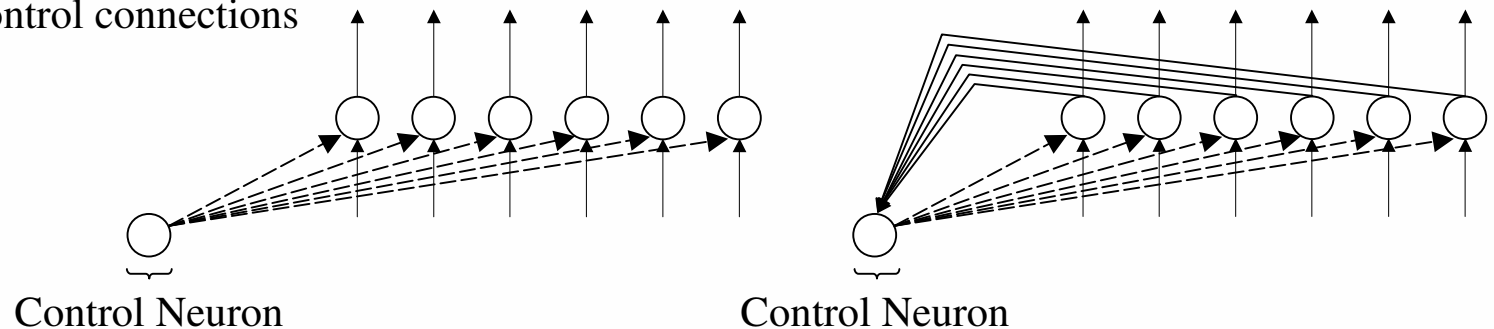
<Neuron> → <enhanced-neuron> (<FunctionSequence>)

$\langle \text{Neuron} \rangle . \text{spec} = \text{substitute}(\langle \text{enhanced-neuron} \rangle . \text{template},$
 $\quad \text{"SUBFUNS"},$
 $\quad \langle \text{FunctionSequence} \rangle . \text{spec})$

Behavioural Variety

- Signal-specific behaviours
 - Enables explicit specification of learning rules and activation rules in single grammar
- Control Structures
 - Behaviour of one neuron or module may explicitly control the behaviour of other structures through appropriate signal-specific connections

- > <perform-learn> connections
- > <activation> connections
- > non-control connections



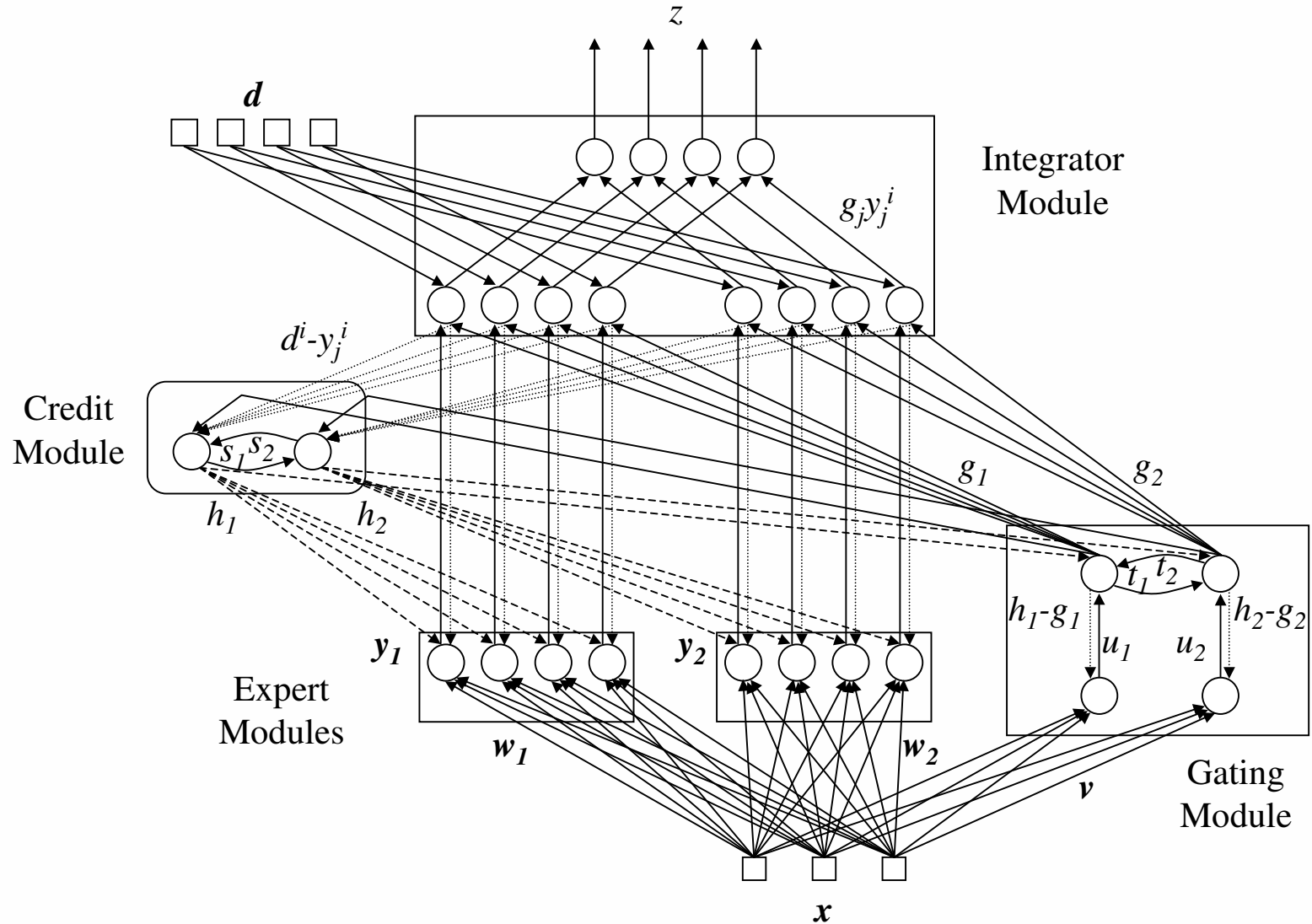
Behavioural Variety

- Share behaviours between modules
 - Share individual neuron behaviours and/or behavioural components (e.g., learning rules)
 - Share control structures
- Propagate behavioural constraints within a module
 - Propagate entire neuron specifications or behavioural components
 - Apply consistent neural control structures
 - Enables varying degrees of homogeneity in neuron behaviour
- Explicit behavioural relationships between modules

Variety of Architectures

- AGE grammars have been developed for several current NN architectures, including
 - Multi-layer Perceptron networks, Back-Propagation networks, Recurrent networks, Mixtures of Experts, CALM networks, Kohonen grids, instar layers, Cellular encoding networks and the topologies of Jacob and Rehder (1993).
- AGE grammars have also been developed for novel architectures, including:
 - Hybrid modular architectures, arbitrary topologies of neurons with highly varied behaviours, and networks with explicit task decompositions.

Example: Mixtures of Experts Architecture



Attribute Grammar Parse Trees as Genetic Encoding

- The context-free parse trees generated from an attribute grammar may be used as a genetic representation
 - Similar to previous work on logic grammar (Wong and Leung, 1997) and definite clause translation grammar (Ross, 2001) representations
- Subtrees exhibit strong semantic identity
- Trees are compact compared to phenotype
- Grammar design promotes modularity of the phenotype (e.g., modular topology)
- Coarse phenotype structure may be represented near root and fine structure near leaves of tree

Lessons Learned

- Generic specification language that is independent of grammar design is difficult.
 - Design of GNML and AGE was valuable exercise in understanding underlying neural principles.
- Generic neuron model was robust to a variety of different neural network models.
- Generic interpreter surprisingly stable due to generality of neuron model used.
 - Generic interpreter not very efficient.
- Manipulation of GNML within grammar possible, but conventions required.
- Sub-grammars useful to explore a wide variety of modular architectures in a single AGE.

Conclusions

- AGE+GNML provides the novel capability to specify both neural topology and neural behaviour within a single representation at many levels of details.
- AGE encompasses several current context-free grammar based representation techniques.
- AGE is a valuable tool for the optimization of current NN models and the systematic development of new NN models.

Backup Slides

Biased Subtree Crossover

- Biased subtree crossover on genes \mathbf{g}_1 and \mathbf{g}_2 :
 1. Find $\mathbf{V} = \mathbf{S}_1 \cap \mathbf{S}_2$
 2. Each member $\nu \in \mathbf{V}$ has associated with it a “likelihood” l_ν of selection (where $l_\nu = \rho_\nu$)
 3. Normalize likelihoods l_ν of all $\nu \in \mathbf{V}$
 4. Probabilistically select $\nu_r \in \mathbf{V}$ as the “reproduction symbol”
 5. Identify all nodes in \mathbf{g}_1 that match ν_r and randomly select one.
 6. Identify all nodes in \mathbf{g}_2 that match ν_r and randomly select one.
 7. Swap the subtrees rooted at the selected nodes.

Biased Subtree Mutation

- Biased subtree mutation on gene \mathbf{g} :
 1. Each member $s_i \in \mathbf{S}_{\mathbf{g}}$ has associated with it a “likelihood” l_i of selection (where $l_i = \rho_{s_i}$)
 2. Normalize likelihoods l_i
 3. Probabilistically select $s_r \in \mathbf{S}_{\mathbf{g}}$ as the “reproduction symbol”
 4. Identify all nodes in \mathbf{g} that match s_r and randomly select one.
 5. Replace the subtree rooted at that node with a randomly created subtree rooted with the same symbol s_r
 - If production probabilities are also used, then weight the random decisions made during the creation of the new subtree as appropriate.

Next Steps: Exploitation of Self-Knowledge Provided by Grammar

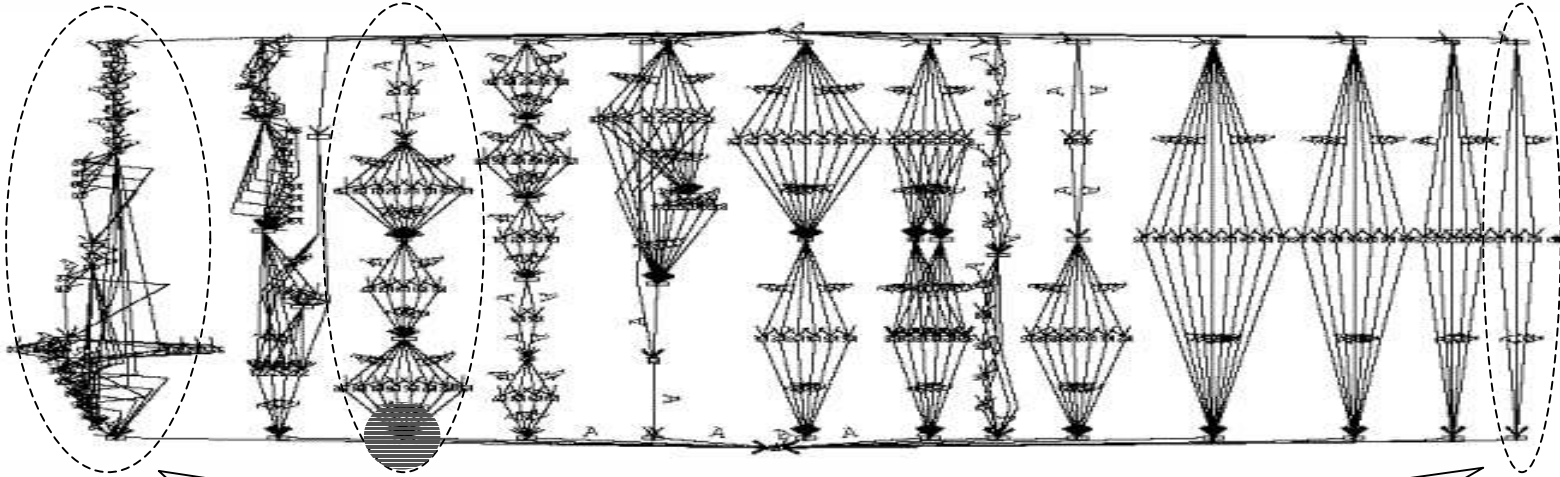
- **Goal:** Improve scalability through the development of mechanisms that exploit information available in the AGE grammar to:
 - Provide “understandable” explanations of system to itself or other systems (and, potentially, to human users).
 - Improve efficiency of learning and computational process.
 - Identify when specific network components are having difficulty learning or exhibiting undesirable behavior.
 - Improve effectiveness of training process.
- Exploit symbolic information available in the parse tree
 - Don’t need to work backwards from the nodes and connections only. Have some high-level internal structure information in the parse tree
 - Use design information to identify

Next Steps: Self-Knowledge for Self-Explanation

- Grammar attributes may be used to automatically generate qualitative and quantitative descriptions of the network (i.e., in addition to GNML specification).
 - “System solves tasks q, r, s before t, u, v.”
 - “System needs values for data elements i1, i3, i4 before it can compute outputs o2, o3, o6.”
- Automated identification of potential inefficiencies in modular design before training begins.
 - “Networks a, b and c do not contribute to final result.”
 - “Networks d and e are performing identical tasks.”
 - “Module X requires significantly more resources.”
- Analysis of the derivation may discover important characteristics of the neural structure.
 - A repeated structure may represent a meaningful high-level computation that has broader application.

Module 3 contains 5 experts.
Expert 1 performs complex mapping. Possible training bottleneck.

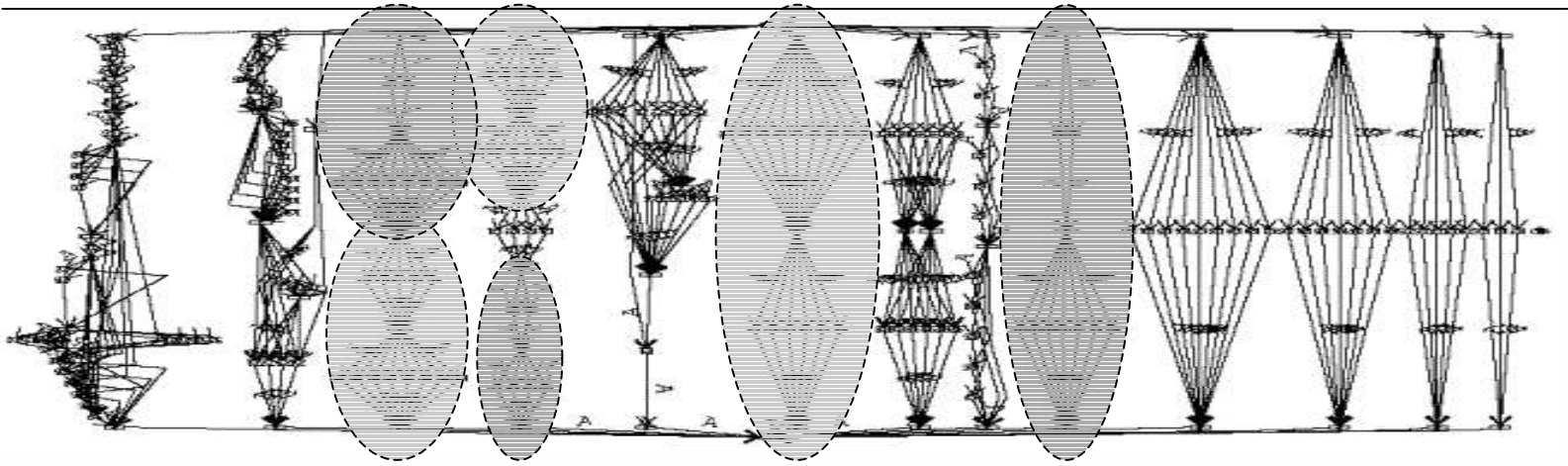
Output 0 requires 8 modules and 73 experts to compute.
Output 1 requires 1 module and 3 experts to compute.
...



Module 1 and Module 13 both solve the same task.
Module 13 requires 90% fewer resources than Module 1.

Repeated structure

Repeated structure



Self-Knowledge for Self-Improved Efficiency

- During training, self-knowledge from grammar may be used and augmented to improve computational and learning efficiency.
- Perform self-diagnostics.
 - Identify poorly performing network experts.
 - Compute confidence in performance of a network expert.
 - Identify sources of error, e.g., “The value provided for input i1 has frequently produced spurious results in output o2.”
- Prune inefficient network structures.
 - Redundant network elements.
 - Network elements contribute minimally to solution.
 - Modules that are highly computation-intensive.
 - Network elements that consistently produce meaningless output.
- Adapt learning parameters.
 - e.g., increase learning rate in networks that are static.
 - e.g., reduce learning rate in modules that cooperate to produce error-prone output element o3.

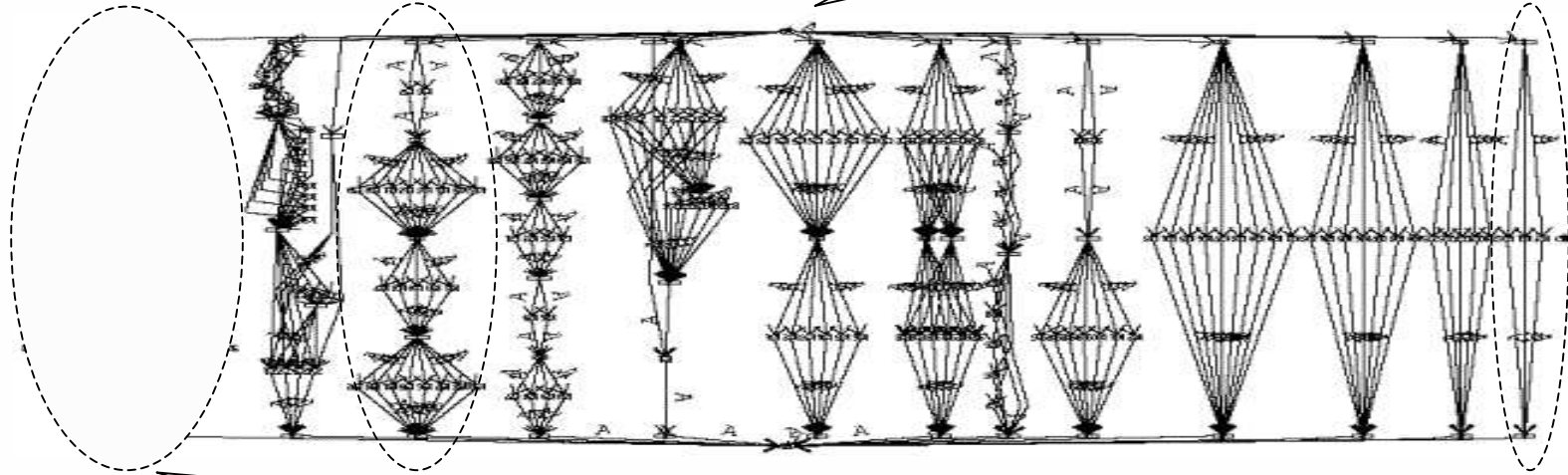
Module 3 exhibits extremely poor learning performance.

□ Increase learning rate of all experts in Module 3.

Output 0 requires 7 modules and 53 experts to compute.

Output 1 requires 1 module and 3 experts to compute.

...



Module 1 performs poorly compared to Module 13.

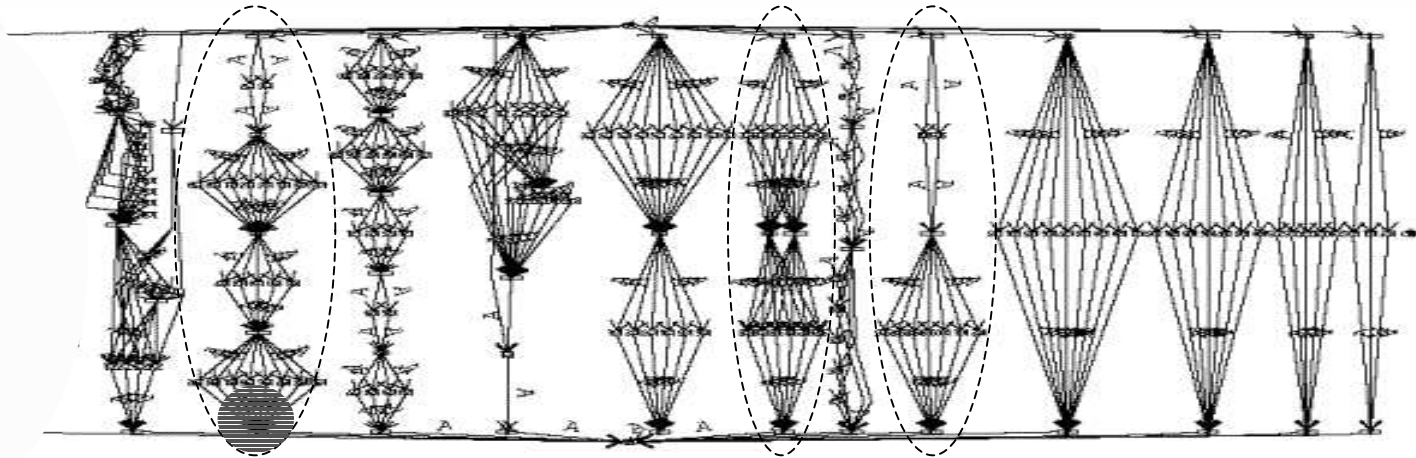
□ Module 1 eliminated to reduce redundancy.

Self-Knowledge for Self-Improved Training Effectiveness

- A key capability required to facilitate training of large networks of complex experts on large problems is **focused training**.
 - In current modular approaches, all network elements are trained with equal bias.
 - However, not all sub-tasks within a network are equally easy and not all expert modules train at the same speed.
- Focused training involves independent training of specific network elements and/or specific data elements for extra trials.
 - Increasingly important as complexity of problem grows.
- During AGE training, dynamic training mechanisms that exploit self-knowledge for focused training may be used.
 - Identify system elements (experts and/or modules) with poor learning and perform focused training upon those elements.
 - Compute confidence measures for all modules, and perform focused training upon low confidence modules.
 - Identify error-prone output elements and perform focused training upon network elements that cooperate on those outputs.

Module 3 still exhibits
extremely poor learning.

Resolve by performing
focused training on Expert 1.



Output \square has very high error.

Modules 7 and 9 cooperate to provide Output \square

Perform focused training on Modules 7 and 9.

Self-Knowledge in Neural Networks: AGE Benefits

- **Improved Introspection:** Novel facilities for understanding generated structures that provide:
 - self-knowledge that is exploited to improve performance.
 - explanations to user before, during and after training.
- Focus is upon generating qualitative descriptions and quantitative measures that may be exploited.
 - Typical architecture is highly modular and task decompositions are more explicit.
- Self-knowledge is extracted early and used to improve learning and efficiency.
- Extraction of self-knowledge from grammar attributes is computationally cheap.
- Additional self-knowledge is discovered and enhanced as learning proceeds, and is used to make training more effective.

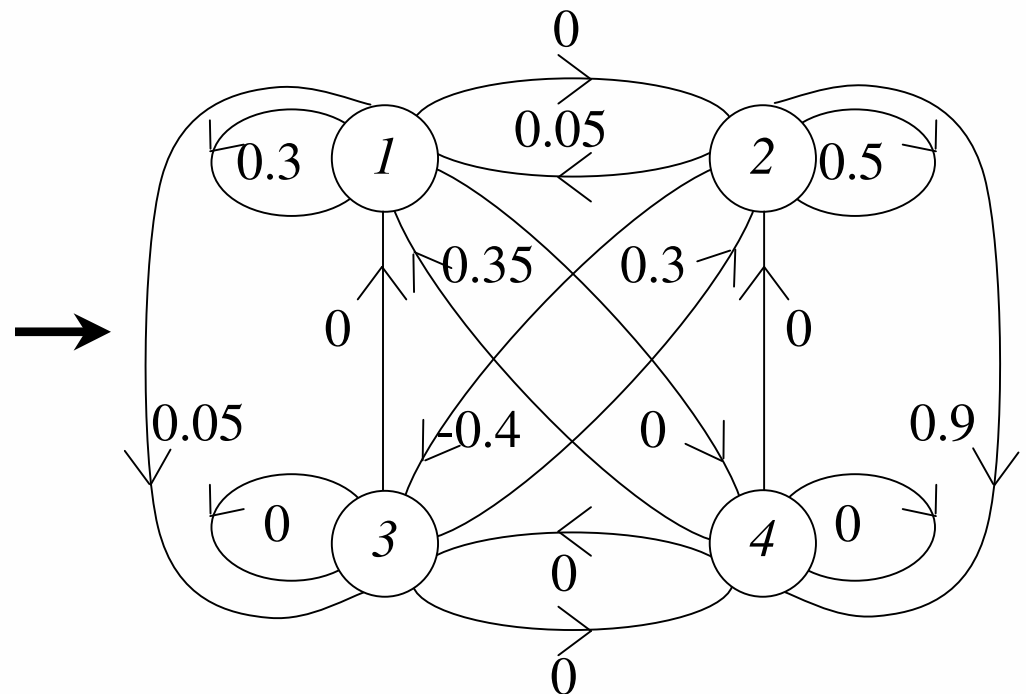
Self-Knowledge in Neural Networks: Prior Work

- Prior work on the analysis of neural networks has focused primarily upon the extraction of rules from classic feed-forward networks. (Andrews et al., 1995)
 - Typical architecture is highly connected, and internal representations are highly distributed.
 - Spatial and functional decompositions are implicit.
 - Extraction of rules from ensembles of neural experts has also been studied. (Wall et al., 2002)
- Two approaches to rule extraction:
 - Derive rules from analysis of internal structure of network.
 - Derive rules from the input/output behavior of network.
- Rule extraction is computationally expensive and performed only once learning is complete.

Background: Direct Encoding

- Exact specification of the weights of a fixed topology
- Many architectural assumptions
- Scales poorly

	<i>1</i>	<i>2</i>	<i>3</i>	<i>4</i>
<i>1</i>	0.3	0	0.05	0
<i>2</i>	0.05	0.5	-0.4	0.9
<i>3</i>	0	0.3	0	0
<i>4</i>	0.35	0	0	0

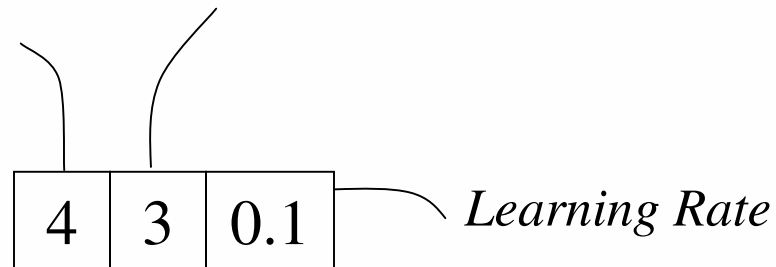


Background: Parametric Encoding

- Specification of high-level architectural parameters
- Restricted set of architectures
- Scales well

*# Nodes in First
Hidden Layer*

*# Nodes in Second
Hidden Layer*




Background: Rule Encoding

- Specification of a set of graph re-write rules that are applied to a start symbol
- Highly varying topologies but strict, homogeneous functional assumptions
- Scalability varies

$$a \rightarrow \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} b \rightarrow \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} c \rightarrow \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \dots p \rightarrow \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

ABBAacpbapba

$$S \rightarrow \begin{bmatrix} A & B \\ B & A \end{bmatrix} \quad A \rightarrow \begin{bmatrix} a & c \\ p & b \end{bmatrix} \quad B \rightarrow \begin{bmatrix} a & p \\ b & a \end{bmatrix}$$



0	0	0	1	0	0	1	1
0	0	0	0	0	0	1	1
1	1	1	0	1	0	0	0
1	1	0	0	0	0	0	0

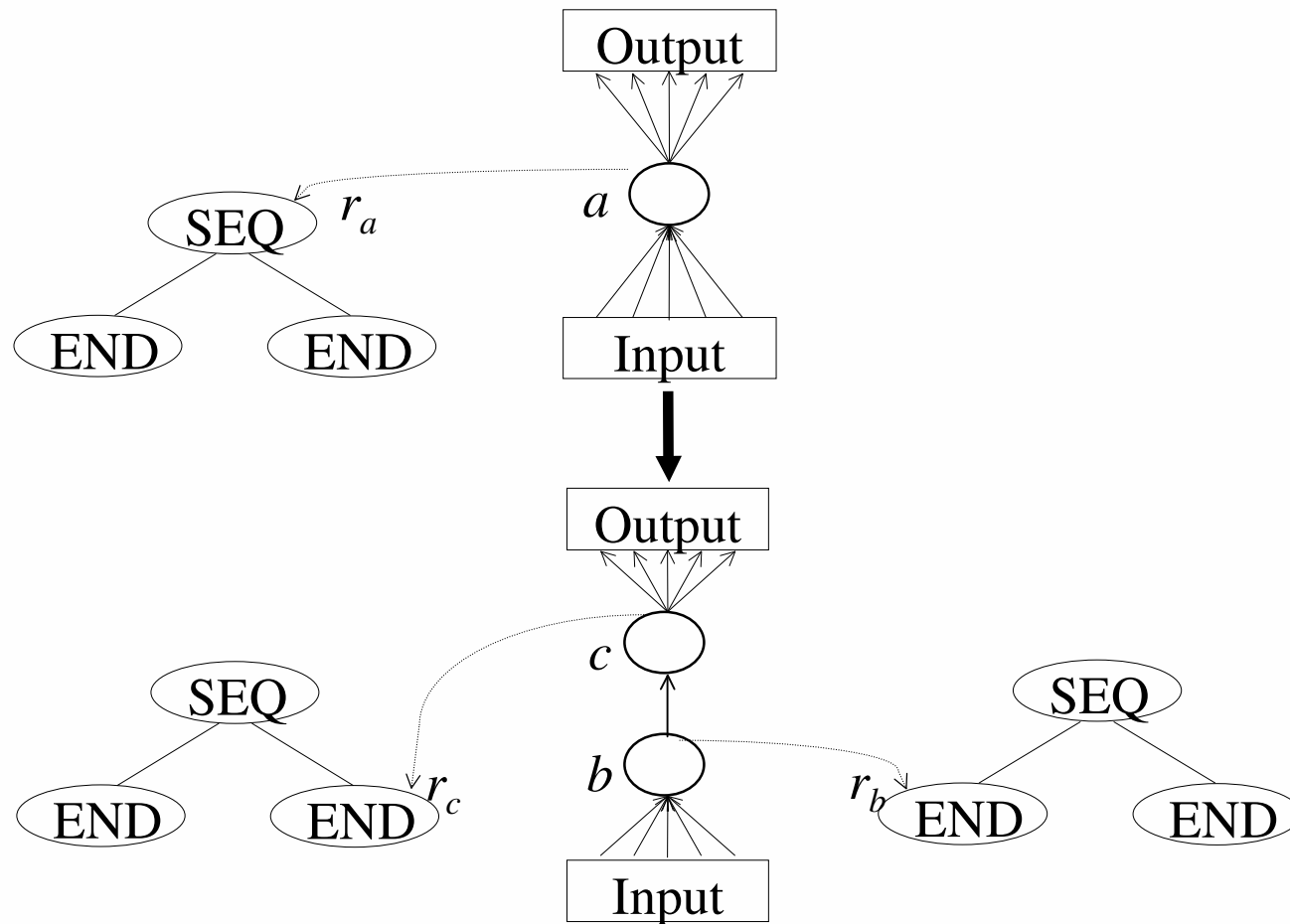
Background: Cellular Encoding

- Given a pre-defined set of cell transformation rules, specifies a sequence rules that are applied to expand a given starting cell into a network
- Gene is a parse tree of rules
- A rule may split a cell into multiple cells and may specify connections between cells
- Every cell owns a copy of the tree and applies a transformation rule indicated by the location of that cell's "reading head"
- Arbitrary topologies, but strict, homogeneous functional assumptions

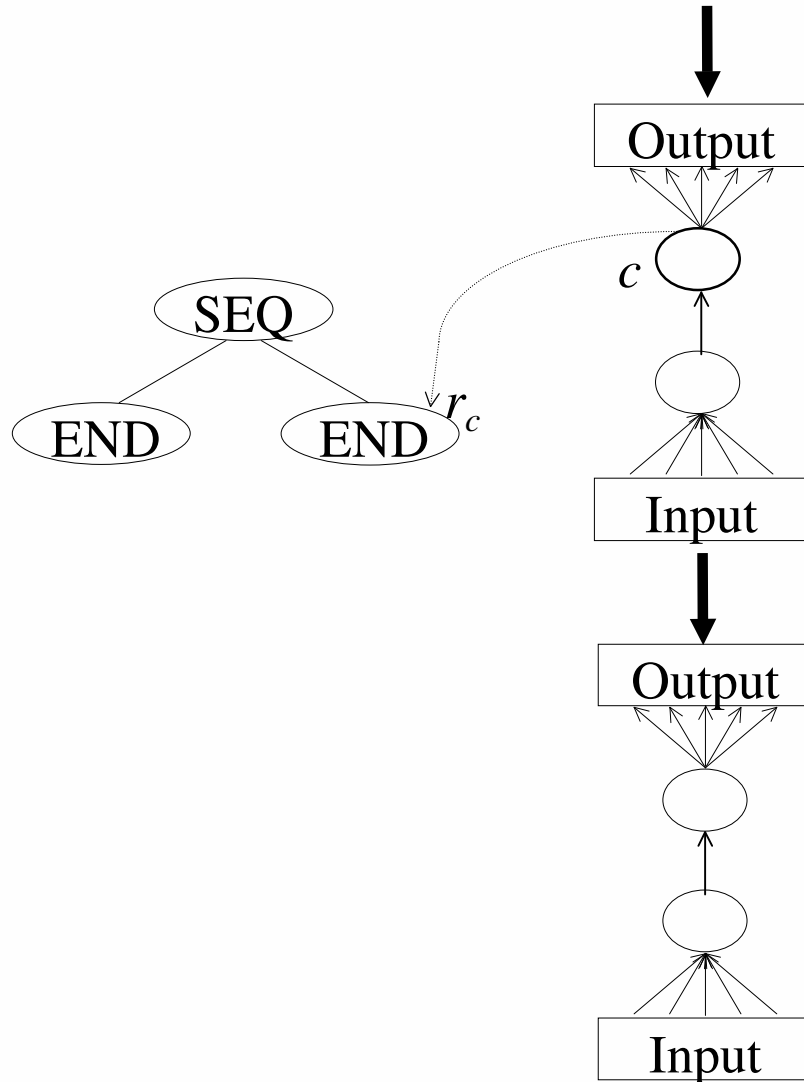
Background: Cellular Encoding

- Basic transformation rules:
 - SEQ:
 - split a cell a into two cells b and c
 - b 's reading head is set to the left child of a 's reading head
 - c 's reading head is set to the right child of a 's reading head
 - b has the same input connections as a
 - c has the same output connections as a
 - a single connection is made from a to b
 - PAR
 - first three steps as above
 - b and c have the same input and output connections as a
 - b and c are not connected
 - END
 - converts a cell into a final neuron that can no longer change

Background: Cellular Encoding



Background: Cellular Encoding



Background: Cellular Encoding

- Cells are evaluated on a FIFO basis.
- Left child is placed on a queue before right child
- Effectively, this is a left-to-right breadth-first traversal of the gene tree
- However, rules may move the reading head to the root (REC) to an arbitrary location in the tree (JMP) or make a cell WAIT to be evaluated by moving from the front to the back of the queue
- These rules produce idiosyncratic tree traversal.
 - Subtrees do not always have a clear structural identity. The same subtree in different locations in the parse tree can produce very different structures due to the use of WAIT, JMP and REC rules

Experimental Results

- Training Set: 20,000 examples of 26 letters from the Machine Learning Repository of the University of California, Irvine.
- Back-propagation networks used
 - Hidden Layers: 1-3
 - Nodes per layer: 5-30, increments of 5
 - Number of possible networks: 258
 - Training: 200 epochs each
 - Performance: 17% to 80%, average 63%
 - Networks ranked by performance from 1 (best) to 233 (worst)

Experimental Results

- Genetic algorithm:
 - Fixed population size: 10
 - Operators: Mutation 10%, Crossover 90%
 - Number of generations: 100
 - Selection method: Fitness proportional, one operation per generation, and Worst-one(s)-out competitive
 - Average number of unique networks examined during course of evolution: 48

Experimental Results

- 5 searches made per method

<u>Method Employed</u>	<u>Average Best Rank</u>
Genetic Search	3.4
Genetic Search with Symbol Reinforcement	1.6
Random Selection of 48 networks	18.8

References

- Andrews R., Diederich J. and Tickle A. (1995). “A Survey and Critique of Techniques For Extracting Rules From Trained Artificial Neural Networks,” *Knowledge Based Systems*, 8, p. 373-389.
- Bengio, S., Bengio, Y., and Cloutier, J. (1994). “Use of genetic programming for the search of a new learning rule for neural networks,” *Proceedings of the First Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence*, p. 324-327.
- Gruau, F. (1995). “Automatic definition of modular neural networks,” *Adaptive Behavior*, 3(2), p. 151-183.
- Happel, B.L.M. and Murre, J.M.J. (1992). “Designing modular neural network architectures using a genetic algorithm,” *Artificial Neural Networks, Vol. 2*. I. Aleksander and J. Taylor (Eds.), North-Holland: Elsevier Science, p. 1215-1218.
- Jacob, C. and Rehder, J. (1993). “Evolution of neural net architectures by a hierarchical grammar-based genetic system,” *Proceedings of the 1993 International Conference on Artificial Neural Networks and Genetic Algorithms*. R. Albrecht, C. Reeves and N.C. Steele (Eds.), Wien: Springer-Verlag, p. 72-79.
- Knuth, D. E. (1968). “The semantics of context-free languages,” *Mathematical Systems Theory*, 2(2), p.127-145.
- Jordan, M.I. and Jacobs, R.A. (1993). *Hierarchical Mixtures of Experts and the EM Algorithm*. Technical Report: A.I. Memo #1440, Artificial Intelligence Laboratory, Massachusetts Institute of Technology.
- Ross, B.J. (2001). “Logic-based genetic programming with Definite Clause Translation Grammars,” *New Generation Computing*, 19(4), p. 313-337.
- Wall, R., Cunningham, P. and Walsh, P. (2002). *Explaining Predictions from a Neural Network Ensemble One at a Time*. Trinity College Dublin Technical Report TCD-CS-2002-21, April.
- Wong, M.L. and Leung, K.S. (1997). “Evolutionary program induction directed by logic grammars,” *Evolutionary Computation*, 5(2), p. 143-180.