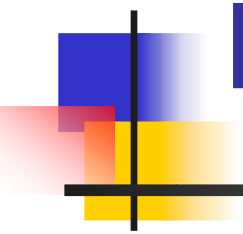
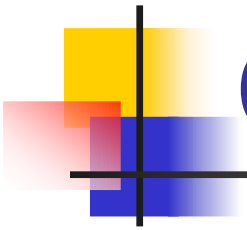


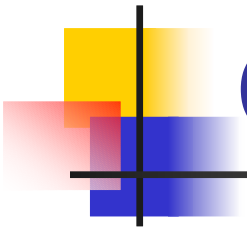
Probabilistic & Machine Learning Applications

Joel Coburn
Ilya Katsnelson
Brad Schumitsch
Jean Suh



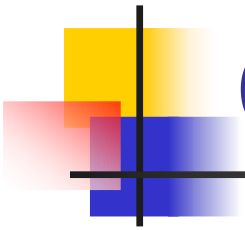
Outline

- Genetic algorithms
- Functionality of learning algorithms
- Characteristics of neural networks
- Available parallelism
- System bottlenecks
- Trade-off analysis



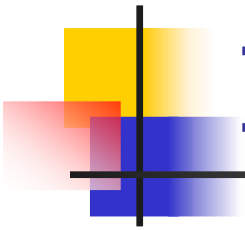
Genetic Algorithms (GE)

- Generally a search procedure that optimizes to some objective
  - Maintains a population of candidate solutions
  - Employs operations inspired by genetics (crossover and mutation) to generate a new population from the previous one
  - Finds the fittest solution candidate
  - Migrates the candidates to generate better “gene pool”
  - Repeats the entire procedure until the specified level of goodness is achieved



Genetic Algorithms (2)

- Used in a large number of scientific and engineering problems and models:
  - Optimization,
  - Automatic programming,
  - VLSI design,
  - Machine learning,
  - Economics,
  - Immune systems,
  - Ecology,
  - Population genetics,
  - Evolution learning and social systems

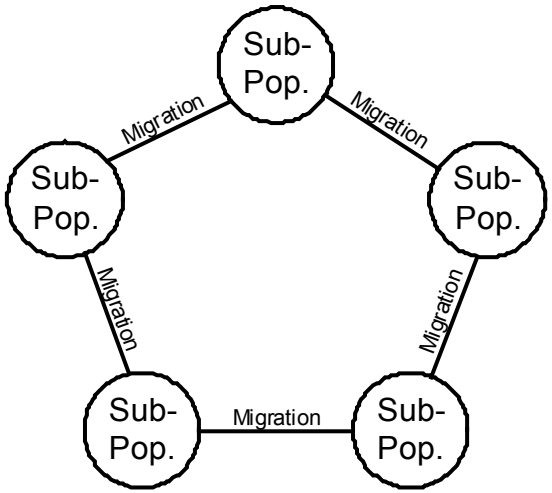


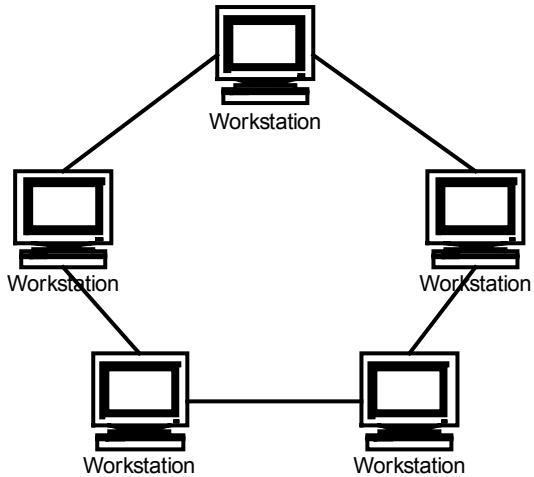
Implementation

- Massively parallel.
  - Most iterations are independent.
  - Several versions of the same candidate solution can be executed in parallel (to collect statistical data) thus providing even more parallelism.
  - Different algorithm models map naturally to a specific HW architecture (see the figure on the next slide).
    - The island model can be readily implemented in distributed memory MIMD computer.
    - The cellular model can be implemented with SIMD computers. MIMD implementations were also performed.

Implementation (2)

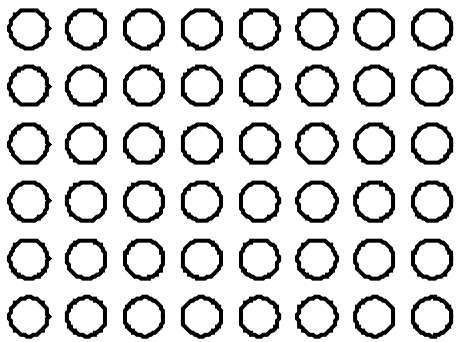
Distributed or Island Model

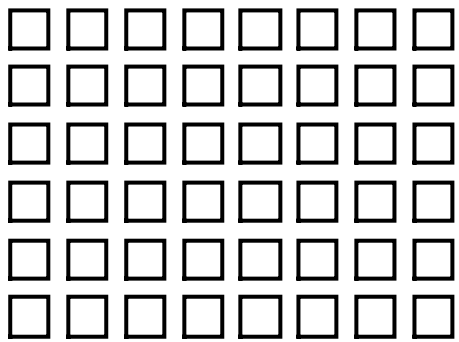




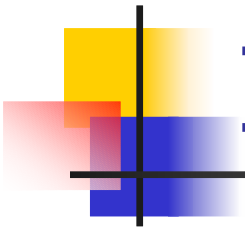
Workstation Cluster

Cellular or Diffusion Model



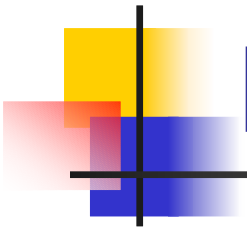


Massively Parallel Computer



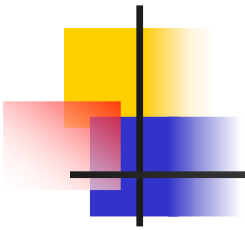
Implementation(3)

- Communication between nodes
  - Almost no communications between independent runs. Can happen only during the migration phases that occur after several generations
  - Determines the number of processors needed to achieve the optimal performance
  - Determines the number of individual candidate solutions that can be put on one island (neighbourhood) for the best performance



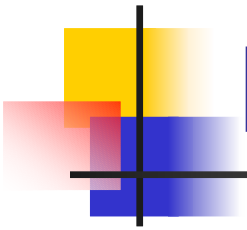
Performance

- Performance largely depends on the target problem and the implementation:
  - Host OS support – how well the host OS supports multithreading and network communication
  - Cache use by the implementation on the host computer
  - Communication between nodes
  - Granularity of the population – number of candidate solutions on a single processing node



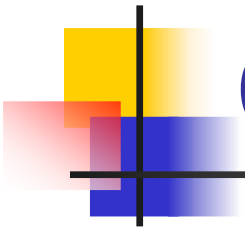
Benchmarks for GAs

- Several Test Suites have been used for long time
  - De Jong test suite (1973)
  - Ackley (1987), Schaffer, Caruana, Eshelman, and Das (1989), Davidor (1991)
- Common problems are used as benchmarks:
  - The Traveling Salesperson
  - The Resource-Constrained Project Scheduling
  - The Flow Shop
- Still in the process of defining



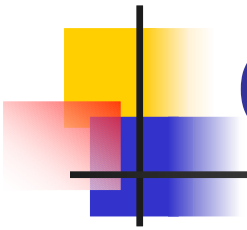
Benchmarks for GAs (2)

- Parameters Used to evaluate GA implementations:
  - The number of evaluations of the function needed to locate the desired optimum
  - Run time needed to locate a solution
  - Speedup – the ratio between the mean execution time on a single processor and the mean execution time on **m** processors
    - Super-linear speedup is achievable
    - Measurement methods are still not standardized



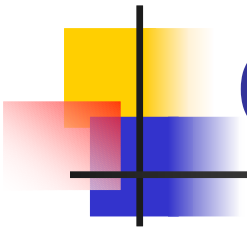
Current Trends

- Current trend is towards using high-level software techniques to implement distributed systems (java, CORBA, sockets, etc.)
  - Only indirect hardware support for parallelism is needed
  - Communication plays significant part in performance
- However, implementations are often done on the heterogeneous sets of platforms



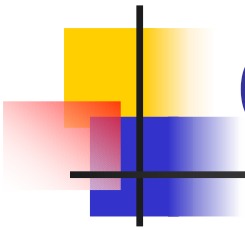
Case Study

- Java-based implementation of parallel distributed GA (alba et. Al.)
  - Studied uniprocessor, homogeneous, and heterogeneous implementations
  - Different host OS support – LINUX, NT, IRIX, DIGITAL
  - Two general GA problems were used (ONEMAX and P-PEAKS) with island configuration



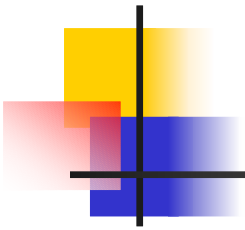
Case Study (2)

- Significant reduction in the number of steps needed to solve the problem when using heterogeneous configurations in comparison with homogeneous.
  - Possible due to the exploitation of the search space at different rates from different heterogeneous machines.
  - Can potentially be a drawback, but generally a good thing because most laboratories have heterogeneous clusters.



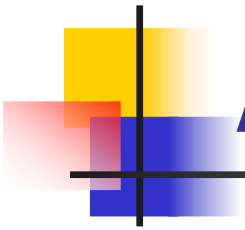
Case Study (3)

- Super-linear speedup for many configurations and problems when using multiprocessor configuration.
  - From hardware viewpoint, when moving from a sequential machine to a parallel one, often not only the CPU power, but other resources such as memory, cache, I/O, etc. increase linearly with the number of processors.
  - Also less overhead for switching from working on one solution to another.



Learning Algorithms

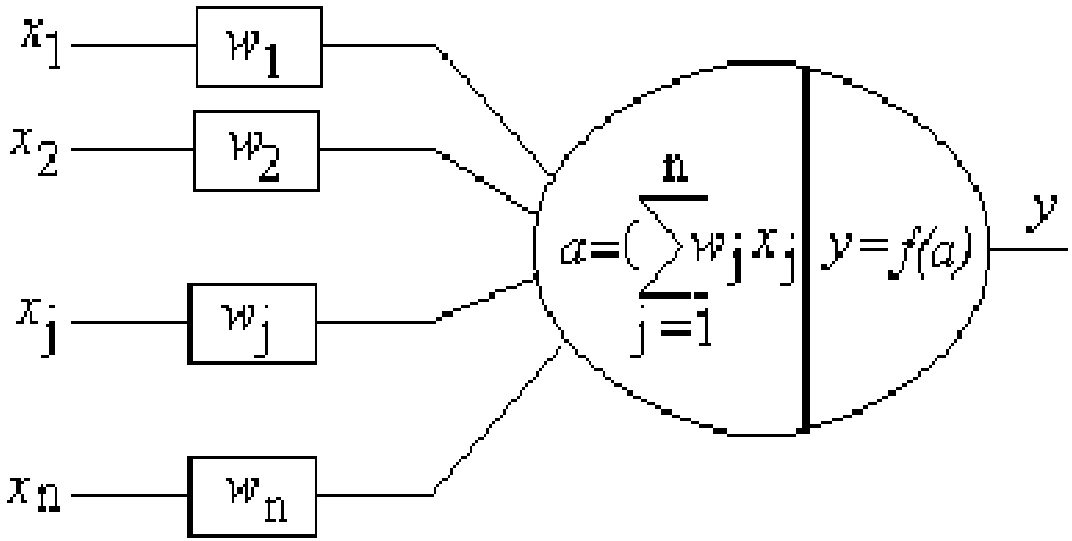
- Consider the set of problems which are easily solved in nature but not by humans
  - No domain knowledge
  - Usually involves a random search for a solution
- Requires massive parallelism
  - An evolving species consists of a large number of individuals competing for survival and reproduction
- Biologically-inspired technique based on learning – Artificial Neural Networks (ANN)



Artificial Neural Networks

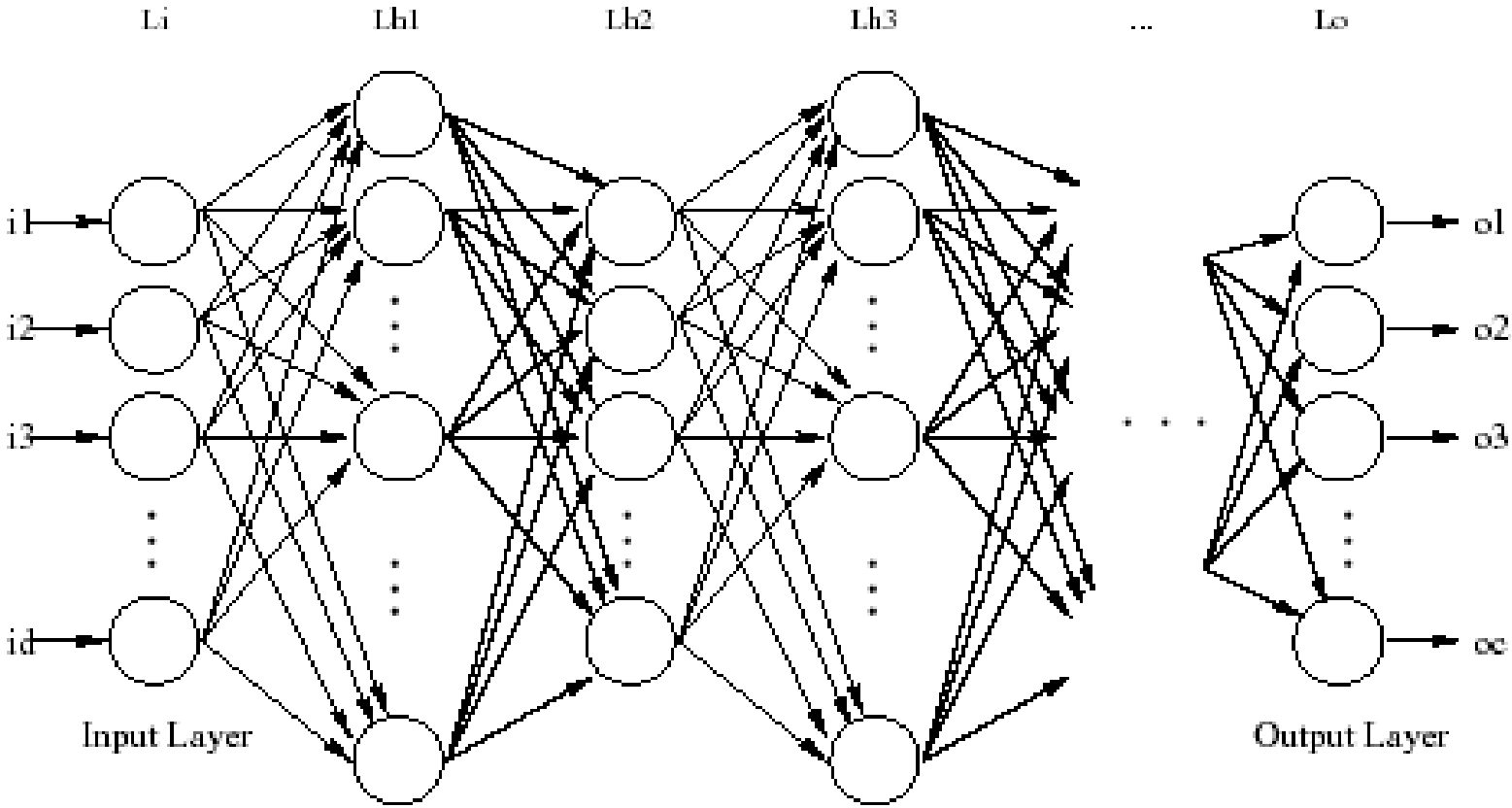
- Neural Networks: non-linear static or dynamical systems that learn to solve problems from examples [Ienne]
- Artificial neuron: accepts several input variables and derives an output from their weighted sum
- To solve interesting problems (handwriting and voice recognition, etc.), we combine many artificial neurons to form an ANN

Artificial Neuron



Activation function is weighted summation of the inputs
Can represent in vector notation as $a=w^T x$

Neural Network



Derivation of Gradient Descent

$$E = \sum_p E_p = \frac{1}{2} \sum_p \sum_j (o_{pj} - a_{pj})^2 \qquad net_j = \sum_i w_{ji} a_i \qquad a_i = f(net_i) = \frac{1}{1 + e^{-net_i}}$$

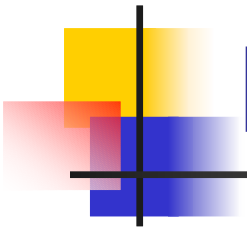
$$\Delta_p w_{ji} \propto -\frac{\partial E_p}{\partial w_{ji}} = -\frac{\partial E_p}{\partial net_{pj}} \frac{\partial net_{pj}}{\partial w_{ji}} \qquad \frac{\partial net_{pj}}{\partial w_{ji}} = \frac{\partial}{\partial w_{ji}} \sum_i w_{ji} a_{pi} = a_{pi}$$

$$\delta_{pj} = -\frac{\partial E_p}{\partial net_{pj}} = -\frac{\partial E_p}{\partial a_{pj}} \frac{\partial a_{pj}}{\partial net_{pj}} \qquad \frac{\partial a_{pj}}{\partial net_{pj}} = f'(net_{pj}) \qquad \frac{\partial E_p}{\partial a_{pj}} = -(o_{pj} - a_{pj})$$

$$\delta_{pj} = (o_{pj} - a_{pj}) f'(net_{pj}) \qquad E^p = \frac{1}{2} \sum_p \sum_j (o_{pj} - a_{pj})^2 \qquad \Delta_p w_{ji} = \epsilon \delta_j^p a_i^p$$

$$\delta_j^p = (o_j^p - a_j^p) f_j'(net_j^p) \qquad f_j'(net_j^p) = a_j^p (1 - a_j^p) \qquad \delta_j^p = f_j'(net_j^p) \sum_n \delta_n^p w_{nj}$$

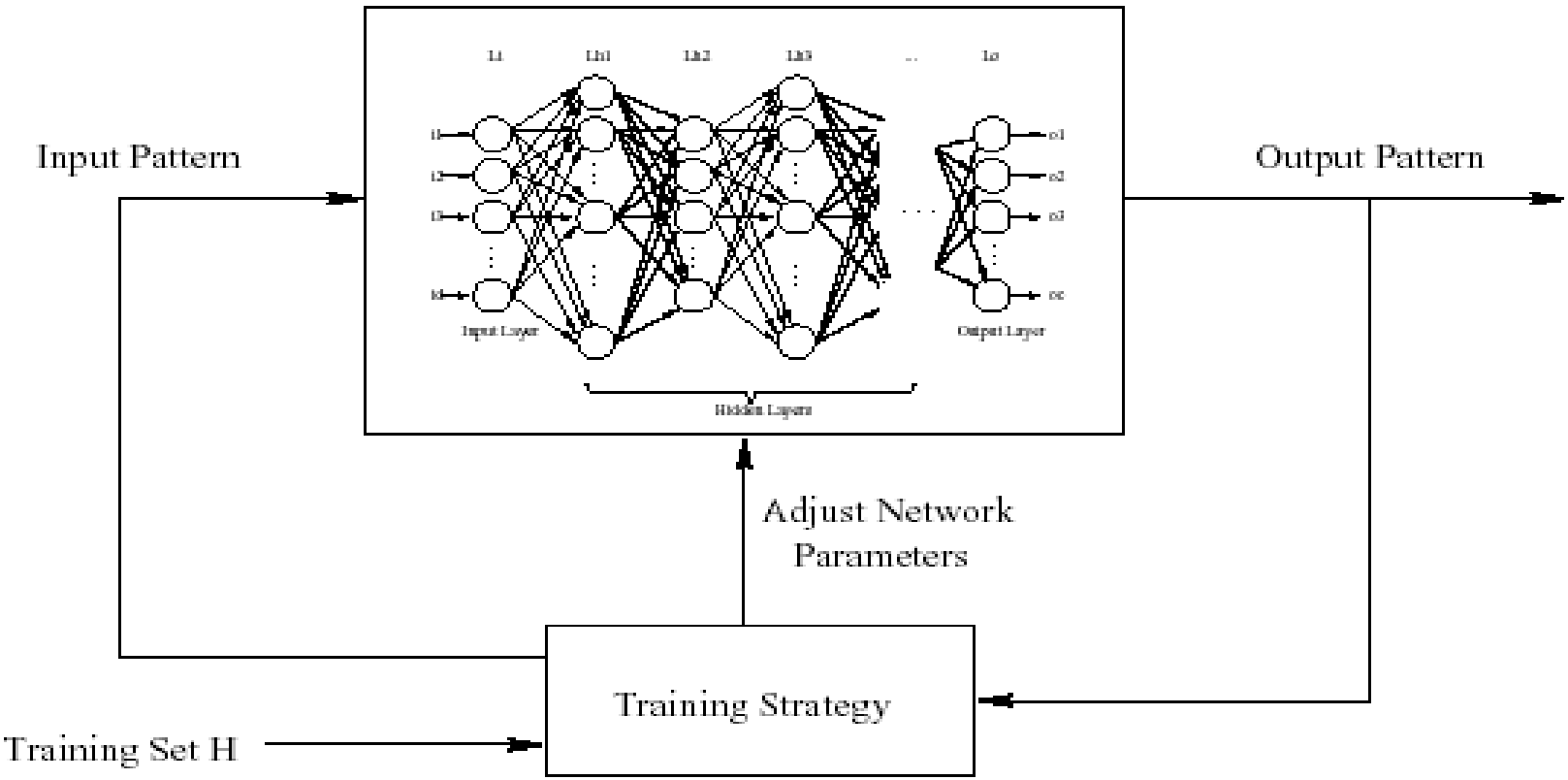
$$a_i = f\left(\sum_{i=0}^{N_k - 1} w_{ij} x_i - \Theta_j \right) \qquad O \le j \le N_k \qquad w_{ji}(n+1) = w_{ji}(n) + \epsilon \delta_j a_i$$



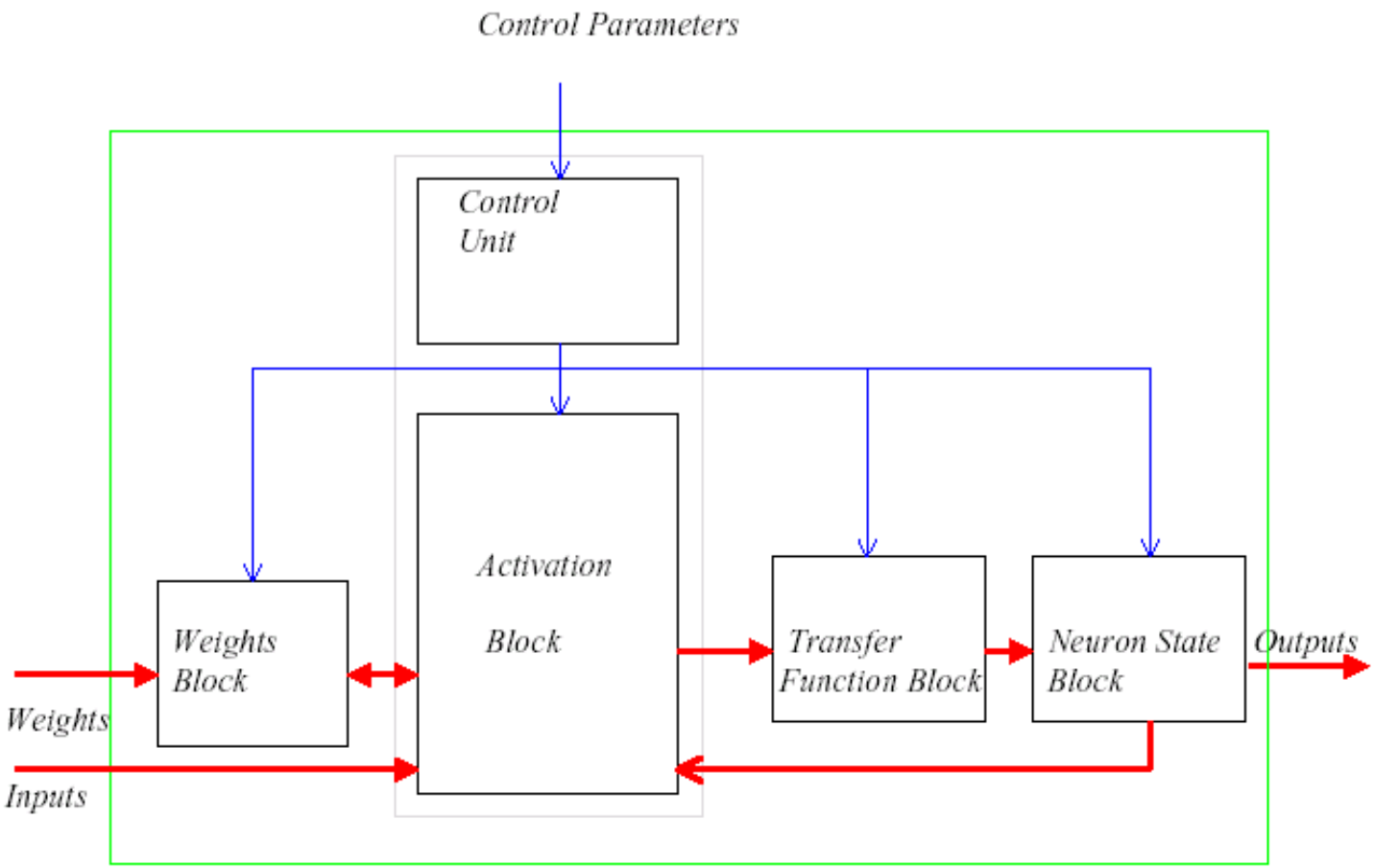
Behaviour During Execution

- Learning Phase
  - Iterate through multiple data sets
  - Feedback loop to provide correction term for weights
- Processing Phase
  - Normal mapping through the network

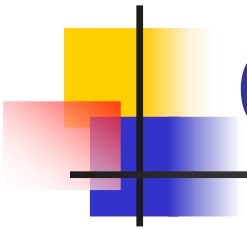
Training Algorithm



Architectural Representation

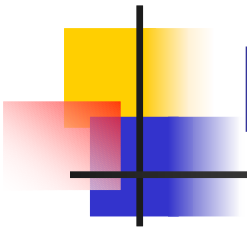


Processing Element (single neuron)



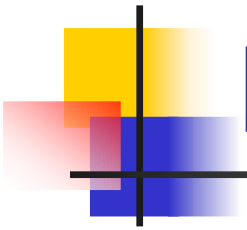
Computational Complexity

- Each input may pass through multiple processing elements depending on the number of layers in the network
- Each processing element performs several operations:
  - An update of activation values in the form of a matrix-vector multiplication
  - Addition to determine final activation value
  - Output function y=f(a)
- Even a small data set yields many computations



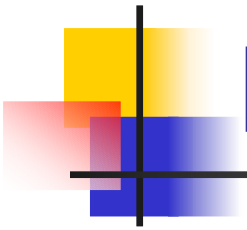
Parallelism

- Significant parallelism available
- Conceptually, a processor per neuron (TLP)
- 3 non-orthogonal ways to exploit it



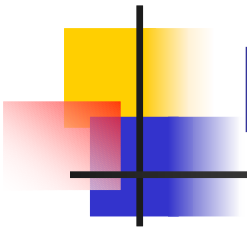
Node Parallelism (1)

- Also neuron parallelism
- One processing element per node
- Set of data inputs(N) from memory or previous layer
  - One output per node
  - N multiplications and N additions
  - O(N) operations per processing element
  - ILP



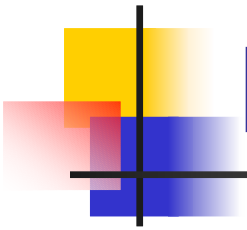
Node Parallelism (2)

- Private storage for weights of each node
  - N weights per neuron
  - All neurons can access their weights at the same time
- Serial data transfer between nodes
  - Only one data transfer between one node to another node of next layer
- Local storage for output data



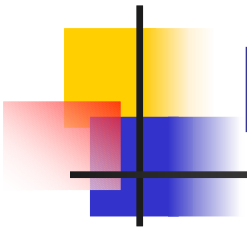
Layer Parallelism (1)

- One processing element per layer
  - Set of data inputs(N) from memory or previous layer
  - Set of data outputs(M) per layer (inputs to the next layer or to output function
  - NM multiplications and NM additions
  - O(NM) operations per processing element
- A buffer to store input and output values for computation and pass on to the next layer



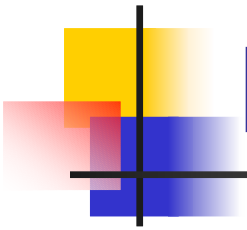
Layer Parallelism (2)

- Central storage for all weights of a layer (shared memory)
  - Duplicated copy of a layer shares the weights
  - If only one copy of layer exists, private storage can be applied
- Serial or parallel data transfer between layers (depends on # of multipliers)
  - If a processing element can do NM multiplications followed by MN additions simultaneously, parallel data transfer can facilitate the performance



Pattern Parallelism (1)

- One processing element per connections
  - Set of data inputs(N) from memory
  - Set of data outputs(M) to memory
  - Set of layers(K), where the i_{th} layer contains L_i neurons
  - (# of multiplications and additions) = $NL_1 + \Sigma L_i L_{i+1} + L_K M$

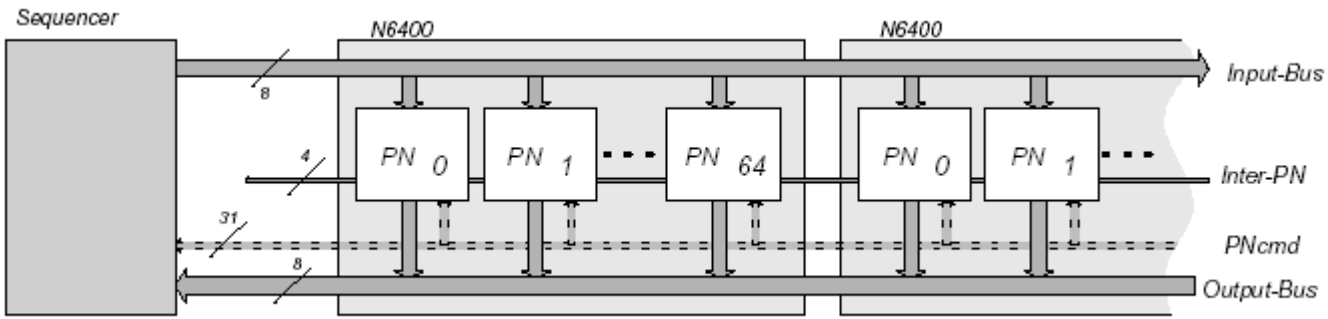


Pattern Parallelism (2)

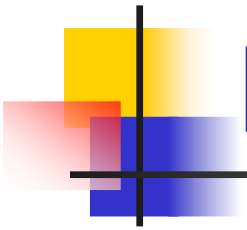
- Each connection of neuron is calculated in parallel
- # Of multipliers = # of connections
- Central storage for all weights of a layer
- Parallel data transfer between layers
- Duplicated network running parallel
  - DLP

Design Example

- Consider CNAPS (Connected Network of Adaptive Processors) from Adaptive Solutions



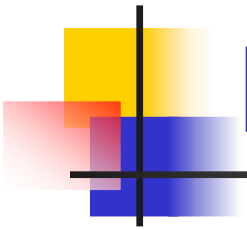
  - Each neuro-chip has 64 processing elements connected to a broadcast bus in SIMD mode
    - Each PN has 4Kbyte on chip SRAM to hold weights



Data Set and Working Set Size

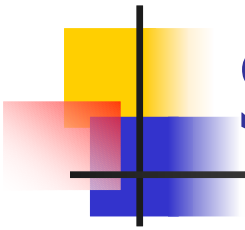
- Most applications use only 10s of inputs
  - Larger networks are rarely used because of the unacceptable learning-time required
  - Could be increased if special-purpose hardware is available
- Working set size is a function of the number of neurons in the system
  - Each neuron is typically operating on several data elements at a time

Arithmetic Operations and Memory Access

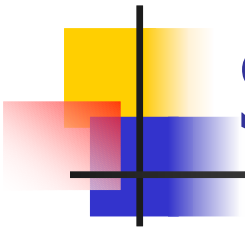


- For the learning or processing phases, the data set will have to be fetched from memory
  - Either one large parallel operation (too many ports) or slower serial access
- Inside each PN:
  - Read from SRAM to find weight for each input – can be direct-mapped to fit data set
  - Multiply and add
  - Ratio of arithmetic/memory operations for each PN is about 2

Bottlenecks on Current Systems

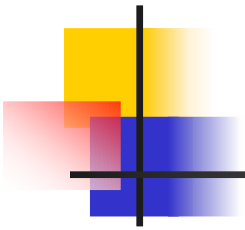


- Communication bandwidth
  - Many interconnections between processing elements
- Cost
  - Approximate to the number of processors required
- Complex programming interfaces
- Power consumption
- Large area



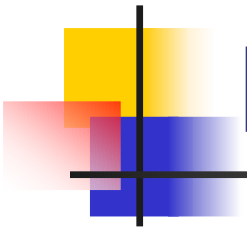
Scaling Trends

- Goal: Reach the performance of biological synaptic networks
  - Energy Gap: digital VLSI technology requires much more energy to implement a synaptic connection (order of 106)
  - Capacity Gap: Storage density is far less than that of biological networks (order of 106)
- Technology scaling will shrink feature sizes and make synaptic cells more compact
- Less arithmetic precision and supply voltage can reduce energy requirements



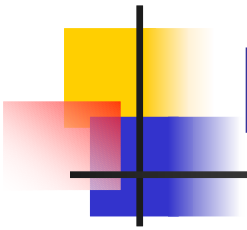
Performance Evaluation

- ANN performance is measured by two metrics:
  - Processing Speed: Multiply and accumulate operations performed in unit time = MCPS (Millions of Connections Per Second)
  - Learning Speed: Rate of weight updates = MCUPS (Millions of Connection Updates Per Second)
- These metrics ignore learning convergence



Benchmarks

- Very scarce, but designs should be tested with as many training data sets as possible
- Neural network benchmarking collections
  - CMU nnbench
  - UCI machine learning databases archive
  - Proben1
  - StatLog data
  - ELENA data



References

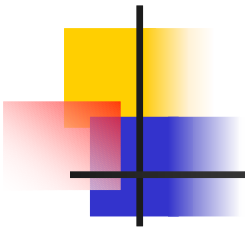
- Aybay, I., Cetinkaya, S., Halici, U., 1996, "Classification of Neural Network Hardware", *Neural Network World*, IDG Co.,Vol 6 No 1.
- Burr, J., 1993, "Digital Neurochip Design," Chapter 8 of *Parallel Digital Implementations of Neural Networks*, H. W. Przytula and V. K. Prasanna, eds., Englewood Cliffs: Prentice Hall.
- Burr, J., 1991, "Energy, Capacity, and Technology Scaling in Digital VLSI Neural Networks", *IEEE International Conference on Computer Design.*

References (2)

- Cornu, T., Ienne, P., 1994, “Performance of digital neuro-computers.”, *Proceedings of the Fourth International Conference on Microelectronics for Neural Networks and Fuzzy Systems*.
- Heemskerk, J.N.H, 1995, “Overview of Neural Hardware.”, later published in his Ph.D. thesis.
- Ienne, P., 1993, “Architectures for Neuro-Computers: Review and Performance Evaluation.”, *EPFL Technical Report 93/21*.

References (3)

- Ienne, P., Kuhn, G., 1995, "Digital Systems for Neural Networks.", *Digital Signal Processing Technology*, SPIE Optical engineering.
- Jahnke, A., Klar, H., Schoenauer, T., 1998, "Digital Neurohardware: Principles and Perspectives.", *Neural Networks in Applications*, Institute of Microelectronics, Technical University of Berlin.



References (4)

- Whitley D, Rana S, Dzubera J, et al. "Evaluating evolutionary algorithms." *Artificial Intelligence*. 85 (1-2): 245-276 Aug 1996.
- Alba E, Nebro AJ, Troya JM. "Heterogeneous computing and parallel genetic algorithms." J*ournal of Parallel and Distributed Computing*. 62 (9): 1362-1385 Sep 2002.
- "Solutions to parallel and distributed computing problems : lessons from biological sciences." Edited by Albert Y. Zomaya, Fikret Ercal, Stephan Olariu. New York : John Wiley, c2001.